9-2005

Modeling Adaptive Middleware and Its Applications to Military Tactical Datalinks

Jason T. Lawson

MODELING ADAPTIVE MIDDLEWARE AND ITS APPLICATIONS TO MILITARY TACTICAL DATALINKS

THESIS

Jason T. Lawson, Captain, USAF

AFIT/GCE/ENG/05-08

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GCE/ENG/05-08

MODELING ADAPTIVE MIDDLEWARE AND ITS APPLICATION TO MILITARY TACTICAL DATALINKS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Jason T. Lawson, BS

Capt, USAF

June 2005

AFIT/GCE/ENG/05-08

MODELING ADAPTIVE MIDDLEWARE AND ITS APPLICATION TO MILITARY TACTICAL DATALINKS

Jason T. Lawson, BS
Capt, USAF

Approved:

_____ //signed//_____ _____
Dr. Richard Raines (Chairman) date

_____ //signed//_____ _____
Dr. Rusty O. Baldwin (Member) date

_____ //signed//_____ _____
Dr. Thomas C. Hartrum (Member) date

Acknowledgments

First, I would like thank God for providing the guidance to help me through this challenging and difficult part of my life. Without God leading my life, I am not sure where I would be today.

I would like to express my sincere appreciation to my faculty advisor, Dr. Richard Raines, for his guidance, patience and support throughout the course of this thesis effort. In addition, I would like to thank Dr. Raines for believing in me enough to allow me to pursue this thesis effort with the utmost freedom, even in the face of the personal challenges that were placed before me. I would also like to thank my committee members, Dr. Rusty Baldwin and Dr. Thomas Hartrum, for their guidance and knowledge. I wish to express special thanks to Dr. Baldwin for playing an integral role in helping me through the toughest period of my life, and for instilling in me a new appreciation for spirituality and the role God plays in my life.

Last, but certainly not least, I would like to thank Mr. Kenneth Littlejohn for his support and continuous encouragement. I couldn't ask for a better sponsor, and I will certainly miss the wide variety of discussions, technical and non-technical, we have had over the past few years. Our non-technical discussions are certainly another major reason why God has become an important part of my life.

Jason T. Lawson

Table of Contents

List of Figures

List of Tables

Abstract

Open systems solutions and techniques have become the de facto standard for achieving interoperability between disparate, large-scale, legacy software systems. A key technology among open systems solutions and techniques is middleware. Middleware, in general, is used to isolate applications from dependencies introduced by hardware, operating systems, and other low-level aspects of system architectures. While middleware approaches are or will be integrated into operational military systems, many open questions exist about the appropriate areas to applying middleware.

Adaptive middleware is middleware that provides an application with a run-time adaptation strategy, based upon system-level interfaces and properties. Adaptive middleware is an example of an active applied research area. Adaptive middleware is being developed and applied to meet the ever-increasing challenges set forth by the next generation of mission-critical distributed real-time and embedded (DRE) systems. The driving force behind many next-generation DRE systems is the establishment of QoS requirements typically associated with workloads that vary dynamically.

The Weapon System Open Architecture (WSOA), an adaptive middleware platform developed by Boeing, is modeled as a part of this research to determine the scalability of the architecture. The WSOA adaptive middleware was previously flight-tested with one tactical node, and the test results represent the performance baseline the architecture. The WSOA adaptive middleware is modeled with 1, 2, 4, 8 and 16 tactical nodes. The results of the modeling and simulation is that the WSOA adaptive middleware

can achieve the performance baseline achieved during the original flight-test, in the cases of 1, 2, and 4 tactical nodes. In addition, the results of the modeling and simulation also demonstrate that the WSOA adaptive middleware cannot achieve the original performance baseline, in the cases of 8 and 16 tactical nodes.

MODELING ADAPTIVE MIDDLEWARE AND ITS APPLICATIONS TO MILITARY TACTICAL DATALINKS

1. Introduction

The Weapon System Open Architecture (WSOA) program was initiated in 1999 by the AFRL, the Defense Advanced Research Projects Agency (DARPA), and the Open Systems Joint Task Force (OS-JTF). The goal of the WSOA program is to develop an open-systems "bridge" between legacy embedded mission systems and off-board command and control (C2) resources [5]. Open system approaches and techniques were used because of their potential to address technical limitations that affect the ability of current systems to prosecute time-sensitive targets (TSTs). These technical limitations include bandwidth of current military tactical datalinks, static resource management, and finite computing resources [5].

The architecture developed under the WSOA program is based in large part upon Bold Stroke, a middleware-centric systems architecture developed by the Boeing Company for Operational Flight Programs (OFPs). The Bold Stroke architecture fosters the development of OFPs across multiple fighter aircraft platforms, using standard, commercial-off-the-shelf (COTS) hardware and software [5]. The WSOA architecture combines the middleware foundation of Bold Stroke, which is based on the Common Object Request Broker Architecture (CORBA) standard, along with a QoS management framework, real-time adaptive resource manager (RTARM) and an adaptive scheduling framework. The aforementioned technologies are combined to support applications that dynamically allocate and manage various system resources in response to changes in the operating environment, while providing guaranteed real-time performance of critical tasks.

1.1 Background

The military tactical datalink that WSOA uses is commonly known as Link-16, as defined in MIL-STD 6016. Link-16 is an encrypted, jam-resistant, nodeless datalink used by terminals compatible with the Joint Tactical Information Distribution System (JTIDS), and supports the TADIL J message catalogue [11]. Nodeless networks can use over several different medium access schemes and Link-16 uses both Time-Division Multiple Access (TDMA) and Code-Division Multiple Access protocols. TDMA assigns Time Slot Blocks (TSBs) to individual assets, while CDMA allocates Link-16 datalink networks, otherwise known as Network Participation Groups (NPGs). Link-16 supports the distribution of a wide range of combat information in near-real time to U.S. combat aircraft and command and control centers [11]. In addition, Link-16 has been fielded by NATO and has seen extensive use in Europe. Information transmitted over Link-16 datalink networks include an integrated air picture with both friendly and hostile aircraft locations, general situation awareness data, and additional data on potential air and ground targets [11]. When encryption and jam-resistance are enabled, the maximum achievable bandwidth of a given Link-16 datalink network is approximately 56 Kbps.

1.2 Research Problem

Modeling and simulating the WSOA architecture to determine its scalability is the principal goal of this research effort. The modeling and simulation tool used to investigate various properties of networking protocols is OPNET®. OPNET® models communication systems of all types and levels of protocols [10]. OPNET® Modeler supports many types of networking technologies to include TDMA communications

of standards-based protocol models, with completely open source code.

The current WSOA architecture supports a single command and control aircraft and a single tactical fighter node. For the purposes of demonstrating the application of new technology, this type of limited experimental setup was sufficient. However, since this technology will eventually transition to existing military systems, the scalability of the WSOA architecture and underlying technology must be established. Specifically, the goal of this study is to estimate the number of tactical fighter nodes that can be supported at varying levels of QoS by a given command and control node. Within the context of this study, support is defined by the requirements set forth by individual tactical fighter nodes with respect to the various data products provided by the command and control aircraft. For example, the Weapon System Officer (WSO) for an F-15E Strike Eagle may define the maximum allowable time for downloading an image to be displayed on the Tactical Situation Display (TSD).

1.3 Hypothesis

The hypothesis of this study is that the QoS management framework, embedded within the WSOA middleware architecture, will allow the command and control aircraft to provide adequate support for at least 16 tactical fighter nodes. As discussed previously, one major goal of this study is to determine an estimated value for n, the maximum number of tactical fighter nodes that can be adequately supported. Furthermore, once $n + 1$ and increasing numbers of tactical fighter nodes are being supported by the command and control aircraft, it is expected that the WSOA architecture will no longer be able to support the total number of tactical fighter nodes. Therefore, the requirements set forth by individual tactical fighter nodes will not be met for various data products provided by the

command and control aircraft. Thus, individual and collective operational capability of tactical fighter nodes will not be realized, resulting in an overall loss of military effectiveness.

1.4 Approach

The general approach taken to investigate the stated hypothesis, and other performance-related metrics, is through the use of a discrete-event simulator. Given that the WSOA architecture consists primarily of various communication protocols, the OPNET® simulation tool is used for building the experimental model and performing all experiments described herein. The OPNET® simulation tool is a discrete-event simulator targeted to simulate various types of network communication systems [21].

Various performance metrics are calculated or measured based upon the simulation results produced by exercising the overall system model. The performance metrics being used are based upon injecting a known workload into the system, in the form of simulated servicing of image requests originating from *n* individual tactical fighter nodes. The effects of this workload will be measured through two metrics: throughput measured in image tiles per second, and the compression level of image tiles that are transmitted.

The metrics will be compared to data collected from the WSOA flight test for purposes of validation and verification, and a performance and scalability analysis will be conducted based upon varying the known workload.

1.5 Summary

The remainder of this document is organized into four chapters. Chapter 2 contains the literature review where background associated with adaptive middleware is presented. The methodology for the experimental phase of this investigation is given in Chapter 3. The analysis of the results and comparison to earlier works follow in Chapter 4. Finally, Chapter 5 provided a summary of the thesis effort and identifies areas of the research to be explored in future research efforts.

2. Literature Review

2.1 Introduction

This chapter provides an overview of pertinent literature relating to adaptive middleware and more specifically, the application of adaptive middleware to military tactical datalinks for the purposes of enabling enhanced communications capabilities. This chapter is organized into six areas, starting with an overview of the Weapon System Open Architecture (WSOA) program, followed by a discussion of current and future military tactical datalinks. Within the context of the WSOA program, and its relationship to current military tactical datalinks, a detailed discussion of the three key components of adaptive middleware is provided, which include quality of service (QoS) management frameworks, adaptive scheduling techniques and dynamic resource management approaches. Finally, this chapter closes with a survey of approaches to modeling adaptive middleware and its associated components, within an environment amenable to studying the performance of packet-switched communications systems.

Open systems approaches and techniques have become the de facto standard for achieving interoperability between disparate, large-scale, legacy software systems [5]. A key technology among open systems approaches and techniques is middleware. The middleware concept was developed based upon recognizing the opportunity to develop and evolve systems through reusable software [24]. Middleware, in general, is used to isolate applications from dependencies introduced by hardware, operating systems, and other low-level aspects of system architectures. Numerous efforts are currently underway to develop and field Operational Flight Programs (OFPs) based upon open systems approaches such as middleware [25]. While middleware approaches are or will be

integrated into operational military systems, many questions exist pertaining to the boundaries of applying middleware.

Adaptive middleware, one such application boundary, is currently an active research topic in the literature. Specifically, adaptive middleware is being developed and applied to meet the ever-increasing challenges set forth by the next generation of mission-critical distributed real-time and embedded (DRE) systems [9]. The driving force behind many next-generation DRE systems is the establishment of QoS requirements, typically associated with workloads that vary dynamically.

In addition, given the distributed nature of these new systems, the varying workloads introduced by them are often serviced by shared resources. As such, achieving QoS requirements in these types of environments requires new adaptive techniques, such as automated reconfiguration, layered resource management, and dynamic scheduling [9]. Combined with middleware, these new adaptive techniques can be encapsulated to introduce application-level awareness of QoS into next-generation DRE systems, without the creation of low-level system dependencies resulting in expensive coupling between various layers of such systems.

2.2 Weapon System Open Architecture (WSOA) Program Overview

The WSOA program was initiated in 1999 by the Air Force Research Laboratory (AFRL), the Defense Advanced Research Projects Agency (DARPA), and the Open Systems Joint Task Force (OS-JTF). The goal of the WSOA program is to develop an open-systems "bridge" between legacy embedded mission systems and off-board command and control (C2) resources [5]. Open system approaches and techniques are

seen as a way to address technical limitations that affect the ability of current systems to prosecute time-sensitive targets (TSTs). Technical limitations include bandwidth of military tactical datalinks, static resource management, and finite computing resources [5].

The architecture developed under the WSOA program is based in large part upon Bold Stroke, a middleware-centric systems architecture developed by the Boeing Company for OFPs [25]. The Bold Stroke architecture fosters the development of OFPs across multiple fighter aircraft platforms, using standard, commercial-off-the-shelf (COTS) hardware and software [5]. The WSOA architecture combines the middleware foundation of Bold Stroke, based on the Common Object Request Broker Architecture (CORBA) standard, along with a QoS management framework, real-time adaptive resource manager (RTARM), and an adaptive scheduling framework. The aforementioned technologies combine to support applications that can dynamically allocate and manage various system resources in response to changes in the operating environment, while providing guaranteed real-time performance of critical tasks.

Since the foundation of the WSOA architecture is middleware, a review of current middleware technologies is in order. Middleware, or more specifically, distributed object computing (DOC) middleware, can be decomposed into the following layers: domain-specific middleware services, common middleware services, distribution middleware, and host infrastructure middleware [24]. Viewing this decomposition from higher to lower layers as in Figure 2-1, it is not altogether different from the OSI Reference Model for network protocols [9]. In addition, there are a number of competing technologies at each of the layers.

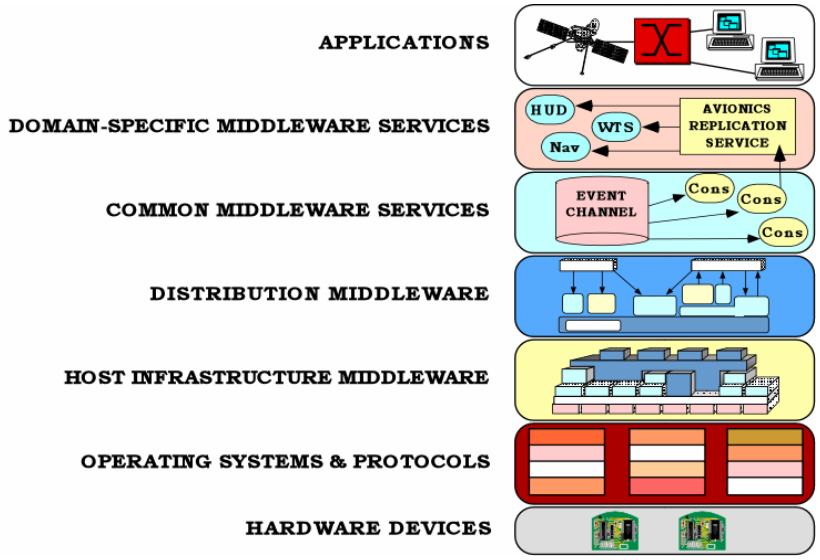

Figure 2-1. Layers of DOC Middleware and Surrounding Context [5]

The lowest layer of DOC middleware is the host infrastructure layer. The purpose of the host infrastructure layer is to encapsulate and enhance native OS communication and concurrency mechanisms to support reusable components and software. Competing technologies at this layer include the Sun Java Virtual Machine [18], .NET [29] which is Microsoft's platform for XML services, and the Adaptive Communication Environment (ACE) [26], a highly portable toolkit developed at Washington University. At this layer of the middleware, the WSOA architecture uses ACE. This choice is dictated by the implementation of the Bold Stroke architecture, which focuses on open commercial standards and technology.

The role of the distribution layer is to define higher-level models for distributed computing, based in large part on reusable components and frameworks that extend the native services of the operating system [24]. Competing technologies include OMG's

CORBA standard, Sun's Java Remote Invocation (RMI) [33], Microsoft's Distributed Component Object Model (DCOM) [3] and an emerging technology known as the Simple Object Access Protocol (SOAP) [27]. At this layer of the middleware, the WSOA architecture implements the CORBA standard.

Next, the function of the common middleware services layer is to augment the distribution layer by defining more abstract domain-independent services that typically are responsible for implementing what is known as the "plumbing code" often required in distributed computing environments [24]. Examples of competing technologies at this layer include OMG's CORBA Common Object Services (CORBAservices) [20], Sun's Enterprise Java Beans (EJB) technology [30], and Microsoft's .NET Web services [29]. At this layer of the middleware, the WSOA architecture implements the CORBAservices.

Finally, the purpose of the domain-specific middleware services is to achieve domain-specific goals and requirements that are not addressed by the lower-level services [24]. A prime example of the technology operating at this layer is the Bold Stroke architecture which defines specific component services to support mission critical functions such as navigation, display management, sensor management, situation awareness, data link management and weapons control. Since the targeted application space is avionics, the WSOA architecture inherently takes advantage of the existing domain-specific services that are implemented as part of the Bold Stroke architecture.

2.3 Relationship of WSOA to Military Tactical Datalinks

The goal of the WSOA program is to develop an open-systems "bridge" between legacy embedded mission systems and off-board command and control (C2) resources,

via military tactical datalinks such as Link-16. To gain insight into the meaning of the term “open-systems bridge”, the relationship of WSOA to military tactical datalinks much be established. This relationship can be clearly established by comparing and contrasting the capabilities of current military tactical datalinks with the capabilities of new applications that are enabled by the development of WSOA.

Although limited, Link-16 does provide combat aircraft and command and control centers a means to exchange data and information. Link-16 is somewhat inflexible since it is based upon an underlying TDMA architecture and relies on the TADIL J message catalogue. WSOA overcomes this limitation by implementing a pluggable protocol through the CORBA communications architecture that has for custom messaging and transport mechanisms [5]. The application-level impact of the pluggable protocol is two-fold. First, implementation of custom messaging, as opposed to reliance on the messages sets defined in the TADIL J catalogues, allows for different types of data to be exchanged between tactical and C2 assets. This benefit is clearly established by a demonstration application developed under the WSOA program. Instead of Link-16 delivering simple track and threat location data, WSOA-enabled applications can deliver richer data sets such as a Virtual Target Folder (VTF). A VTF has descriptive information regarding the target, an index of available imagery via thumbnail images, designated critical point locations, and information concerning threats in the vicinity of the target [5].

Second, custom messaging and transport mechanisms allow more efficient use of bandwidth. This has also been shown by a demonstration application developed under the WSOA program. When a user received a VTF and clicks on an image thumbnail, a request for a larger version of the image submitted. During the download of the larger

image, measures of QoS and resource utilization are monitored to adapt the process of downloading [5]. Simple adaptations include increasing or decreasing the level of compression for individual image tiles based upon whether the previous image tile is behind schedule, on schedule or ahead of schedule [5].

To increase the capability and flexibility introduced by WSOA within Link-16 datalink networks, enhancements and improvements to Link-16 are needed. One promising enhancement to Link-16 is known as Dynamic Networking System for Link–16 (DNS-16) [7]. DNS-16 consists of a three-layer protocol implemented on top of the current Link-16 physical layer. These three layers consist of the Link Monitor-16 (LMON-16), the Unified Slot Allocation Protocol-16 (USAP-16) [35], and the Smart Information Management Systems-16 (SIMS-16) [7]. A hierarchical view of layers is provided in Figure 2-2. To use this new protocol, a proxy is introduced. Proxies provide dynamic networking capability without requiring the upgrade of all Link-16 terminals. Platforms with dynamic networking capability act as proxies for platforms with unmodified terminals [7]. By not upgrading the entire inventory of Link-16 platforms, a dynamic network capability can be achieved at a reduced cost and impact on the warfighter.

LMON-16 provides an interface between the Link-16 terminal and the higher layers by monitoring traffic flow through the terminal itself. Specifically, the LMON-16 layer extracts messages, such as Precise Position, Location, and Identification (PPLI) messages, from the stream and use the information to establish a new dynamic network NPG. In addition, bootstrap messages generated by other dynamic terminals are decoded, and communication tables are constructed in an effort to ensure contention-free

communication [22].

USAP-16 layer provides a set of protocols enabling the network to distribute a common picture of the current operational network to itself [22]. The USAP protocols achieve this by monitoring the RF environment, allocating channel resources on demand based upon a heuristic function, and automatically detecting and resolving contention that results from changes in connectivity. The underlying USAP protocols have been previously developed and demonstrated as part of Soldier Phone, a separate program that supports a multi-net TDMA network architecture [2]. USAP protocols enable contention-free slot assignment within a multi-net TDMA network architecture [22].

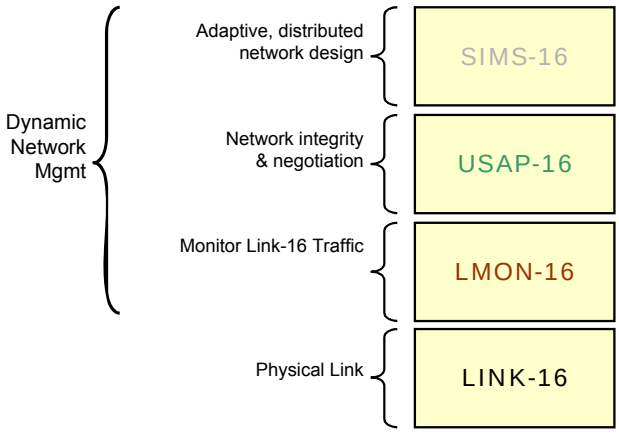

Figure 2-2. DNS-16 Layered Approach to Dynamic Networking [7]

SIMS-16 assigns TSBs to NPGs, making transmit assignments and negotiating proxy assignments [22]. SIMS-16 automatically associates a dynamic terminal with each legacy terminal to serve as its proxy to the USAP-16 datalink network. While any dynamic terminal should be able to serve as a proxy, dynamic terminals serving in an

operational C2 role, such as E2C or AWACS, are preferred over other dynamic terminals [22]. The purpose of a proxy is to recognize terminals without dynamic capability in the vicinity and execute the USAP-16 protocols for them to obtain network bandwidth. After obtaining the required bandwidth, the proxy terminal sends the legacy terminals the appropriate messages to reconfigure those units as necessary to integrate them into the USAP-16 datalink network [22]. In the future, additional functionality may be incorporated into this layer.

2.4 Quality of Service Management Frameworks

Adaptive military applications can be included in the WSOA architecture, in large part, due to the QoS management framework incorporated into the Bold Stroke middleware. As defined by Schantz [23], Quality of Service (QoS) activities improve and control network resources to achieve a certain level of service. In the broadest sense, QoS involves the multitude of properties beyond the application specific functional behavior of a particular distributed application [23]. Examples include performance characteristics, dependability, behavior and adaptability under various changing environments, and security. Other significant QoS activities include specification, negotiation, enforcement, detection, notification, and reconfiguration and adaptation [23]. Each of these processes will be discussed in the following sections.

One QoS management framework is known as the Quality Objects (QuO) framework. The QuO framework supports QoS at the CORBA layer [36]. Specifically, the QuO framework solves current issues in the development of DRE systems including ignoring system properties associated with different environments and platforms, the

difficulty programmers encounter when dealing with WAN-level properties associated with DRE systems, the large barrier to entry regarding the development of minimally adaptive DRE systems, and the inability of programmers to create strongly adaptive systems with cross-platform implementations [36]. Some of these issues are due in large part to the current lack of information regarding such systems, and the lack of maturity concerning associated technology.

The QuO framework provides solutions to these issues in several ways [36]. First, the QuO framework defines system properties as first class entities, and integrates knowledge of these properties so the application can be aware of and handle changes in the operating environment. Second, the QuO framework reduces the variance of system properties via masking, so that programmers can deal with a relatively invariant subset of system properties. Third, the QuO framework exposes key design decisions of a given object's implementation and use to help the application reconfigure dynamically. Finally, the QuO framework supports the reuse of various QuO architectural components at different points in the lifecycle of the application.

QoS management starts with a connection. A connection is a boundary where expected usage patterns and QoS requirements between client and server objects can be negotiated [36]. Delegate object(s) on the client are created to abstract and manage the communication occurring across the connection defined between the client delegate object(s) and the remote server object(s). Once a connection is established, an associated client delegate object(s) is created and bound to a remote server object(s), the definition and negotiation of QoS regions can begin.

A QoS region can be classified into one of two levels of system conditions [36].

First, a negotiated region is a region defined in terms of both the client and server object usage based upon the system conditions the objects attempt to operate in. Typically, a given client delegate object will support a number of negotiated regions. In addition to negotiated regions, reality regions are defined as the actual QoS associated with the interaction of the client and server objects, as measured by the QoS of the runtime system. The adaptive nature of the QuO framework is encapsulated in the specification of handler routines that execute based upon transitions that occur in either the negotiated or reality regions. Handler routines allow the application on the client side to make decisions regarding the usefulness of compensatory actions, or to modify the original QoS requirements of the application.

Adaptivity implies the existence of multiple behaviors that can potentially occur during the execution of DRE systems that implement the QuO framework. For instance, applications can complete tasks later than expected either through tolerating finishing a task later or rescheduling a task for execution at a future time. Another adaptive behavior modifies the work that an application does. Applications may accomplish less work than expected, which can mean greater errors, lower data resolutions, etc. Adaptive behavior concerns the substitution of alternate mechanisms that possess different system properties. Alternate mechanisms include any type of resource not utilized under normal system operating conditions, for example a compression algorithm, used to compress data when throughput exceeds bandwidth limitations [36].

The QuO framework also supports a number of binding times, referred to as commitment epochs [36]. Commitment epochs are established at definition, connection, negotiation, and invocation times. At definition time, QoS regions are defined and bound

to various handlers to create different adaptive behaviors. Typically, this is accomplished via a description language targeted for QuO, and referred to as QDL. At connection time, adaptive behavior is created by instantiated constructs such as delegate objects that can bind the shape of QoS structures enumerated at definition time. During negotiation, bounds are defined that the client delegate object and server object attempt to operate within.

To resolve the second issue, reducing the variance in system properties, three separate steps are taken. First, existing sources of variance are masked through the layering of delegate objects. An example of this masking, within the context of WSOA, is the system-level delegate object that is layered on top of other delegate objects which are monitoring the loading of the processor, the download time for the current image, etc. From a system-level perspective, the sources of variance are masked by the main delegate, which produces an aggregate assessment of overall system QoS state. Second, system knowledge is brought together from different sources. These sources consist primarily of members of the system design team, such as the client designer, object designer, ORB designer. Finally, the designers of the system must ensure that delegate objects are sufficiently complex to handle system conditions as first class objects.

Variance in system properties can occur during routine operation. Systems that support QoS management must be able to mask this variance at different levels in the system, since the information required to recognize this variance is available at different times and at different places. Each layer in the QoS management framework tries to maintain the QoS provided to higher levels by masking changing system conditions within negotiated levels of defined QoS regions. When system conditions change such

that masking is no longer effective, a handler routine passes this information to a higher layer that can adapt to the changing conditions within its masking range [36]. This may result in both layers attempting to change policies, or other simple modifications, to adapt to the new system conditions. When simple modifications are not successful, a change in expectations is realized, which results in the renegotiation of the boundaries of the layer corresponding to the original QoS region. Figure 2-3 depicts a typical scenario where changing system conditions or properties are masked.

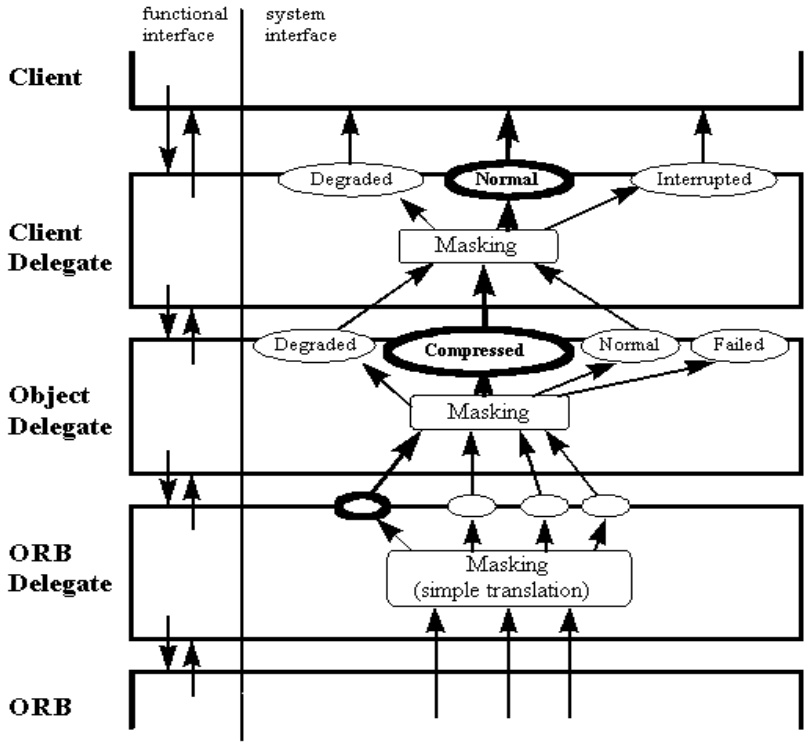

Figure 2-3. Masking System Properties

Integration of system knowledge from different sources is a key process in

reducing the variance of system properties. Sources for this information include the client designer, object designer, QuO designer, ORB designer, and operations staff. Each source can provided different types of information. For instance, the client designer is keenly aware of the need for a delegate object to renegotiate QoS regions. A second example is the operations staff. The operations staff is responsible for knowledge of resource availability, resource access permissions, and administrative domains.

The QuO framework also addresses the third issue, exposing key design decisions of a given object's implementation and use, specifically to provide an application with assistance in reconfiguring dynamically. While many complex software systems can operate effectively based solely on layered abstractions that only expose functional interfaces, DRE systems cannot operate effectively in this type of environment. DRE systems have grown to staggering levels of complexity, with a wide range of resource and usage patterns, and components of DRE systems are required to service a wide range of clients. Thus, a single implementation of a component in a DRE system is not adequate to meet the demands of all possible clients. Open implementation techniques [13] allow system designers to expose key performance and reliability design decisions associated with components and objects. These key design decisions and other usage pattern information of a given component or object, can be abstracted and specified as implementation meta-data [36]. This meta-data is specified separately from the functional aspects of the component or object. Thus, an architecture or framework based upon this meta-data allows a system to reason about itself and adapt to changes occurring within relevant system properties.

The QuO framework specifies separate meta-data using of its Quality Description

Language (QDL). QDL is made up of several independent description languages that specify system property meta-data: the Contract Description Language (CDL), the Resource Description Language (RDL), and the Structure Description Language (SDL). The CDL defines expected usage patterns and QoS requirements for a given connection to an object typically located on a server. The RDL defines the physical resources used by an object. The SDL defines the internal design of an object and quantifies how a given object consumes resources that are allocated to it.

Finally, the Quo framework resolves the fourth issue, the reuse of various architectural components, by introducing new steps in the design process normally associated with developing software within object-oriented frameworks such as CORBA. The overall design process for developing CORBA components and objects is modified to include the role of a QoS designer. In addition, formal and reusable contracts are developed using CDL. This adds another step to the CORBA design process, and likewise introduces additional steps in the design processes for other object-oriented software architectures.

Listing 1 is an example of the structure of a typical contract that contains negotiated QoS regions, from a hypothetical screen-saver application. Specifically, the key elements of the listing are the definition of the contract regions which are defined through the Allocated and Free constructs in the ScreenSaver contract. Within both constructs, the client_expections and object_expectations objects capture the regions of transition for the application, i.e. in terms of throughput and accuracy. Using the Allocated and Free constructs, the appropriate callback methods are executed to force the transition between QoS regions, when changes in the values for throughput and accuracy reach a predetermined boundary.

```
// Forward declarations for classes used in the connection's
// parameters.
interface ScreenSaver_client_callback;
interface ScreenSaver_negotiated_region;
interface ScreenSaver_client_expectations;

connection invScreenSaver(
         // 3 Parameters required for every QDL connection
         // for client_callback
         in ScreenSaver_client_callback cl_call,
         // for client_expectations
         in ScreenSaver_client_expectations cl_exp,
         // for object_expectations
         out ScreenSaver_object_expectations ob_exp,
         // Parameters specific to this connection, which can be used in   //
         predicates for negotiated and reality regions.
         in double max_invoc m_p_s,
         in double max_idle s    ) is

         client_callback interface ScreenSaver_client_callback
         object_callback interface ScreenSaver_object_callback
         client_expectations interface ScreenSaver_client_expectations
         object_expectations interface ScreenSaver_object_expectations

// Meta-level interfaces
contract ScreenSaver is            // CDL negotiated regions are

    Allocated:
       when client_expectations.throughput > 0 m_p_s and
       when client_expectations.throughput <= max_invoc  m_p_s and
       when object_expectations.capacity >= max_invoc m_p_s
    Free:
       when client_expectations.throughput == 0 m_p_s and
       when object_expectations.capacity == 0 m_p_s
    transition callbacks are
       Allocated -> Free:
                  object_callback->client_asleep()
       Free -> Allocated:
                  object_callback->client_awake()
                  client_calllback->now_allocated()
    end transition callbacks

end negotiated regions

reality regions for Allocated are separate
reality regions for Free are separate

end contract ScreenSaver   // CDL

// RDL, SDL, etc. go here
end connection invScreenSaver
```

Listing 1. CDL for ScreenSaver Negotiated Regions

```
separate reality regions for ScreenSaver::Allocated:
       Normal:
          when.QuO_condition.measured_throughput > 0 m_p_s and
          when QuO_condition.measured_throughput <= max_invoc m_p_s and
          when QuO_condition.measured_capacity >= max_invoc m_p_s and
          when QuO_condition.measured_idleness <= max_idle secs

       Insufficient_resources:
          when QuO_condition.measured_capacity < max_invoc m_p_s

       Client_overlimit:
          when QuO_condition.measured_throughput > max_invoc m_p_s

       Client_asleep:
          when QuO_condition.measured_idleness > max_idle sec

       // Precedences tell which reality regions are chosen if more than // one
       predicate is true
       precedence Normal, Client_asleep, Client_overlimit, No_resources
       transitions callbacks are
         Normal -> Insufficient_resources:
              // Warn the client that there isn't enough capacity, even
              // though we're in negotiated region Allocated and thus
              // there is supposed to be capacity.
              client_callback->warn_no_resources()
              // Tell the object to allocate more capacity (or lower its //
              expectations)
              object_callback->allocate_capacity(max_invoc)
         Insufficient_resources -> Normal:
            // Let the client know that it doesn't have to hold its
              // breath any more
              client_callback->warn_enough_resources()
          any -> Client_overlimit:
            // Let the client know it is exceeding its negotiated
            // promise
            client_callback->warn_overlimit(max_invoc)
          any -> Client_asleep:
            // Let both the object and the client know that the client
            // has gone asleep. One or both may reset their expectations
            // (e.g., the client's throughput or the object's capacity),
            //  which could cause a renegotiation.
            client_callback -> warn_sleeping()
            object_callback -> client_asleep()
       end transition callbacks
end separate reality regions ScreenSaver::Allocated
```

Listing 2. CDL for Reality Regions for ScreenSaver Negotiated Region Allocated

Listing 2 provides another example of the structure of a contract that contains QoS region transitions and the associated callback methods, for a hypothetical screen-saver application.

To streamline the process of creating delegate and server objects, automated methods and techniques have been developed to generate objects and software necessary

to build the infrastructure for these new system calls and routines. Although the generation of many delegate and server objects is automated, a number of modifications to these objects are likely to be required if a given contract is being reused across a given software architecture. For instance, client callback routines will likely require re-implementation, or multiple implementations, to deal with changing system conditions.

2.5 Adaptive Scheduling Techniques

Many DRE systems, and other real-time systems, have historically employed static scheduling techniques to enforce deterministic execution of the system, and other real-time performance requirements [16]. This type of scheduling discipline does not provide the flexibility required for a given application to adapt and reconfigure when system conditions change, which in turn affects the overall QoS of the system. Thus, dynamic scheduling methods and techniques that allow systems the flexibility to respond to changes in QoS are needed. It is important to note that QuO only specifies the actions to be taken to manage changes in the system that result in changes in QoS. Other mechanisms, such as dynamic scheduling, are required so the system can react and adapt to changes in the operating environment. As will be discussed later, other mechanisms are also required to allow dynamic and real-time monitoring of resources, the results of which are interpreted by management frameworks such as QuO.

Static scheduling techniques suffer from the following limitations: inefficient handling of non-periodic processing, utilization penalty for non-harmonic periods, and inflexible handling of invocation-to-invocation variation in resource requirements. Static

scheduling handles non-periodic processing inefficiently because such disciplines must treat non-periodic processing as periodic processing that occurs at its maximum possible rate, which typically does not occur in practice. Static scheduling implicitly enforces a phasing penalty for non-harmonic periods. This penalty occurs because tasks with non-harmonic periods introduce unscheduled gaps of time Thus, attaining CPU usage close to 100% is not achievable. Static scheduling also does not allow for flexible handling of resources on an invocation-to-invocation basis. Static scheduling enforces a worst-case allocation of resources, producing a similar type of inflexibility as encountered in non-periodic processing [15].

Dynamic scheduling strategies do not suffer the limitations described previously. Unfortunately, dynamic scheduling strategies mitigate these limitations through increased overhead. In DRE systems additional overhead may introduce other unfavorable conditions. For example, dynamic scheduling strategies can behave non-deterministically under heavy loading conditions. Thus, a careful trade-off must be made when considering the use of dynamic scheduling strategies. Two dynamic scheduling strategies explored under the WSOA architecture, and other avionics applications, are Earliest Deadline First (EDF) and Minimum Latency First (MLF).

EDF [14, 17] gives highest priority to the task with the earliest deadline. A major limitation of EDF scheduling is that the task with the earliest deadline is executed without the probability of meeting its deadline. For instance, a task that requires more time to complete than is actually available prior to reaching its deadline will still be dispatched by the EDF algorithm. A more efficient use of processing resources would be to execute a task with a later deadline that can finish prior to its deadline being reached.

MLF [28] is a scheduling technique that refines the EDF scheduling discipline by accounting for execution time. MLF dispatches an operation or task whose laxity is least. Laxity is defined as the time-to-deadline minus the remaining execution time [15]. Thus, this type of scheduling strategy will detect when an operation or task will not meet its deadline, and then reevaluate the current schedule of operations or tasks.

2.6 Real-Time Adaptive Resource Management Techniques

The Real-Time Adaptive Resource Management (RTARM) system [4], is the methodology that the WSOA architecture uses to dynamically manage and monitor system resources. RTARM supports a number of services that are useful to DRE systems, to include end-to-end QoS negotiation, QoS adaptation, real-time monitoring and hierarchical QoS feedback adaptation. RTARM supports management and monitoring of systems resources, along with network resource management via integration of the NetEx resource management system [4].

Specifically, RTARM uses a hierarchical resource management architecture that provides integrated management over different types of resources. This resource management architecture is recursive, in addition to being structured in a hierarchical fashion. System and network resources are controlled by Service Managers (SMs), which are themselves controlled by higher-level service managers. Figure 2-4 shows a sample RTARM hierarchy consisting of a CPU SM, a network SM and two high-level SMs, to provide integrated resource management capability. Several benefits are realized from utilizing such a hierarchical and recursive resource management strategy. Services with complex QoS requirements and representations are easier to

implement on top of uniform basic services for resource management [4]. An additional benefit of this type of architecture is it allows an application to interact based upon a richer representation of QoS. A drawback of this type of hierarchical approach is the distance between top-level and base-level SMs. If the number of intermediate SMs is large or causes measurable amounts of latency, applications with time-sensitive functionality may or may not be able to implement this type of QoS management framework.

A typical Service Manager is made up of the following functions: Negotiator, Translator, Allocator, Adapter, Scheduler, Enactor, Monitor, Detector and Feedback Adapter [4]. The Negotiator brokers contract admission and can delegate responsibilities to other components. The Translator is used to translate high-level QoS into low-level physical representations. The Allocator is directly responsible for the allocation and release of individual resources. The Adapter performs resource allocation/release depending upon the current state of the QoS contract. The Scheduler determines whether the allocation of resources and the predicted change in system QoS are feasible. The Enactor enforces changes in application-level QoS or other measures of status. The Monitor continuously watches all the associated applications and passes any status information, to include QoS usage, onto the detector. The Detector uses the information passed to it from the Monitor, and detects changes in the operation of a given application. The Feedback Adapter invokes corrective action for a given application when its runtime status, to include QoS, changes significantly.

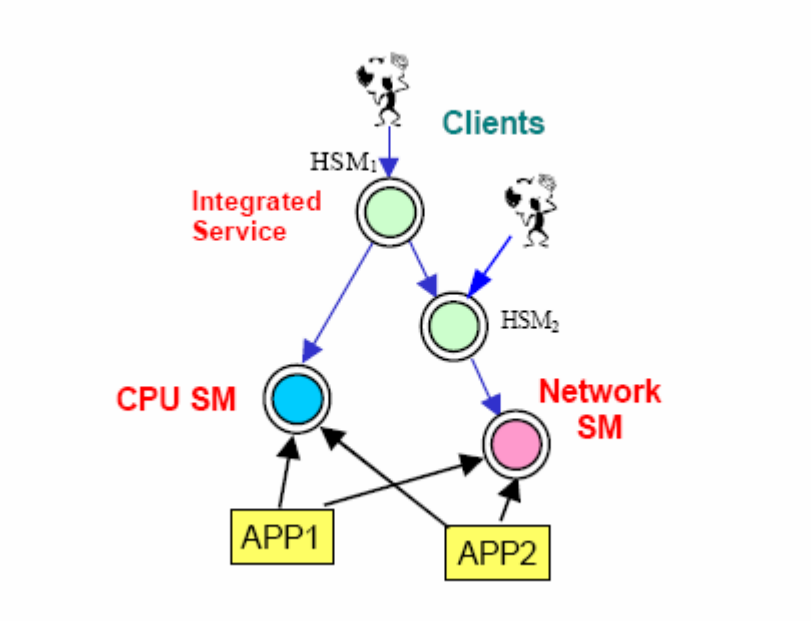

Figure 2-4. Sample RTARM Hierarchy [4]

2.7 Modeling the WSOA Architecture with OPNET®

Modeling and simulating the WSOA architecture to determine its scalability is the principle goal of this research effort. The modeling and simulation tool used to investigate various properties of networking protocols is OPNET®. OPNET® models communication systems of all types and levels of protocols [10]. OPNET® Modeler provides capability and support for simulating many types of networking technologies to include TDMA/CDMA communications networks such as Link-16. In addition, OPNET® Modeler has a comprehensive library of standards-based protocol models, with completely open source code.

Researchers and students at the University of Arizona have used the OPNET® Modeler package to conduct initial research and work into developing OPNET® models

of the CORBA architecture [8]. This research group has modeled the twelve-step process that encapsulates CORBA object communications, to include: client invocation, client data marshalling, client send, server receipt, server data unmarshalling, server upcall, server return, server data marshalling, server send, client receipt, client data unmarshalling, and client return. This research effort also explored using the OPNET® Modeler package to model dynamic invocation, to simulate the Internet Inter-ORB Protocol [19], and to model the CORBA binding operation and naming service [8]. This research is relevant to the effort described here because the CORBA models developed as a part of that research can be utilized as a basis for constructing an adaptive middleware model, as described previously.

2.8 Summary

The literature review in this chapter presents progressively more detailed descriptions of adaptive middleware, and the application of adaptive middleware to military tactical datalinks for the purposes of enabling enhanced communication capabilities. After briefly introducing the Weapon System Open Architecture (WSOA) program, a section is presented on a discussion concerning current and future military tactical datalinks. Next, a detailed discussion of the three key components of adaptive middleware is provided. Specifically, quality of service (QoS) management frameworks, adaptive scheduling techniques and dynamic resource management approaches are all described in detail. This chapter concludes with an overview of approaches to modeling adaptive middleware, and its associated components, within an environment amenable to studying the performance of packet-switched communications systems.

3. Methodology

3.1 Introduction

This chapter describes the methodology used in this effort. The goals of the thesis are presented, followed by the hypothesis. This is followed by a description of the approach and methods used to design the simulation, including performance metrics, system parameters, experimental design and implementation details. Finally, a discussion of the validation and verification associated with experiment is given.

3.2 Goals and Hypothesis

The goal of this study is to determine the maximum number of tactical fighter nodes that can be supported, at varying levels of QoS, by a given command and control node. Within the context of this study, the term adequately is defined by the requirements set forth by individual tactical fighter nodes with respect to the various data products provided by the command and control aircraft. For example, the Weapon System Officer (WSO) for an F-15E Strike Eagle may define the maximum allowable time for downloading an image to be displayed on his or her Tactical Situation Display (TSD).

The hypothesis of this study is that the QoS management framework, embedded within the WSOA middleware architecture, will allow the command and control aircraft to provide adequate support for n tactical fighter nodes. As discussed previously, one major goal of this study is to determine an estimated value for n, the maximum number of tactical fighter nodes that can be adequately supported. Furthermore, once $n + 1$ and increasing numbers of tactical fighter nodes are being supported by the command and control aircraft, it is expected that the WSOA architecture will no longer be able to support the total number of tactical fighter nodes. Therefore, the requirements set forth by

individual tactical fighter nodes will not be met for various data products provided by the command and control aircraft. Thus, individual and collective operational capability of tactical fighter nodes will not be realized, resulting in an overall loss of military effectiveness.

3.3 Approach

The approach used to investigate the stated hypothesis, and other performance-related metrics, is through the use of a discrete-event simulator. Given that the WSOA architecture consists primarily of various communication protocols, the OPNET® network simulation tool is used for building the experimental model and performing all experiments described herein. The OPNET® simulation tool is a discrete-event simulator used to simulate various types of network communication systems [21].

Various performance metrics, as described below and in Chapter 4, are calculated or measured based upon the simulation results produced by exercising the overall system model. The performance metrics are gathered after injecting a known workload into the system in the form of simulated image requests originating from *n* individual tactical fighter nodes. The effects of this workload will be measured through two metrics: throughput measured in image tiles per second, and the compression level of image tiles that are transmitted.

The metrics are compared to data collected from WSOA flight tests for purposes of validation and verification, and a performance and scalability analysis is conducted based upon varying the known workload.

3.4 System Boundaries

The System Under Test (SUT) is the WSOA architecture. Shown in Figure 3-1 is the WSOA architecture, and interfaces the following components: Joint Tactical Information Distribution System (JTIDS) terminals, and the Link-16 communications protocol. The WSOA architecture includes the CORBA-based middleware, the Quality Object (QuO) QoS management framework, RTARM framework, the dynamic scheduling framework (not depicted in Figure 3-1), and the various portions of the WSOA Time-Sensitive Targets (TST) application. The Component Under Test (CUT) is the adaptive middleware, which includes the CORBA-compliant Object Request Brokers (ORBexpress and TAO ORB), the Pluggable Protocols, the QuO Quality of Service Management framework, and the Adaptive Resource Mgmt framework (RTARM).

This study is limited to investigating the scalability of the WSOA architecture within the context of a single Network Participation Group (NPG) as defined by MIL-STD 6016 (Link-16). A NGP is the basic channel used for communication across a Link-16 network. Simulating a single command and control node and multiple tactical fighter nodes is an implicit limitation set forth by the context of a single NPG. This assumes that the typical number of tactical fighter nodes operating on a single NPG will not saturate the capability of the WSOA architecture, although the possibility exists that the results of the experiments will prove that such an assumption is invalid.

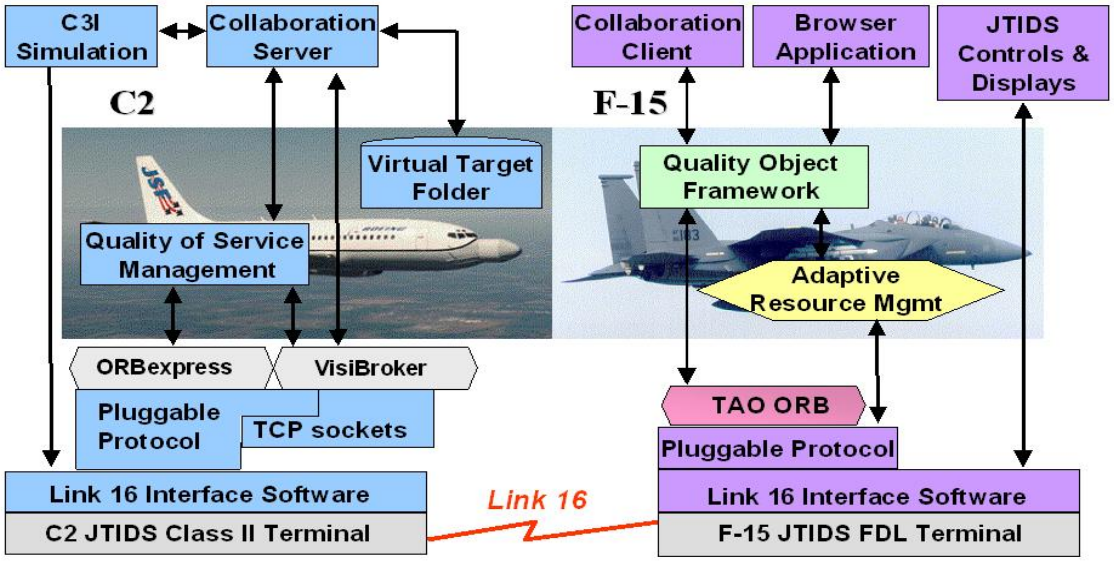

Figure 3-1. WSOA Application and Architecture

3.5 System Services

The basic service provided by the WSOA architecture is the delivery of command and control data in real-time to strategic and tactical military assets. The basic services provided by the WSOA architecture are similar to those provided by the Link-16 communication protocol. Link-16 is currently fielded to support the distribution of a wide range of combat information in near-real time to U.S. combat aircraft and command and control centers [11]. For airborne military assets, examples of command and control data typically include an integrated air picture with friendly and hostile aircraft locations,

general situation awareness data, and additional data on potential air and ground targets [11]. This information is typically displayed on a heads-up display (HUD) or a Tactical Situation Display (TSD).

The difference that exists in services provided by the WSOA architecture, as opposed to the services provided by Link-16, lies in the richness of the data that can be delivered and the additional flexibility in accessing this set of richer data. Instead of transmitting general situational awareness data and information, the WSOA architecture enables applications that can communicate with rich data sets, such as the Virtual Target Folder (VTF). The VTF is made up of thumbnail and full-size imagery, a 9-line briefing, and other descriptive information about the target, and threats in the vicinity of the target. Thumbnail images are used to select and download full-size images from the command and control node [5].

Another implicit service provided by the WSOA architecture is the management of QoS. Management of QoS is handled transparently by the WSOA architecture via monitoring the download of the VTF and associated imagery [5]. The WSOA architecture supports adaptation of the overarching application based upon QoS requirements implemented in the form of QuO contracts [3]. When the specified terms of the QuO contracts are not being achieved, the WSOA architecture can modify the compression level of imagery tiles being downloaded, and thus can support altering the size of image tiles being downloaded.

In summary, there are numerous potential outcomes of the services provided by the WSOA architecture. Given that QoS management is a basic service of the WSOA architecture, one potential outcome is that VTF imagery tiles are transmitted at various

compression levels, corresponding to the current level of QoS supported by the WSOA architecture (i.e., imagery tiles being delivered on time, early or late). Another potential outcome is that VTF imagery tiles are all transmitted at one compression level. This corresponds to either a lack of dynamic QoS management being provided by the WSOA architecture, or an overall time limit for image download that is long enough to accommodate sending all the VTF images at the same compression level.

3.6 Performance Metrics

One primary metric of concern is throughput, measured in image tiles per second. This metric is calculated based upon the number of VTF image tiles per second that are measured in transit across the simulated Link-16 network. This performance metric will be impacted by the ability of the WSOA architecture to adapt to changes in the load placed on the Link-16 network.

Another primary metric of concern is end-to-end image delay time. An overall time limit is set for each tactical node to receive a full 512 x 512 pixel image. Typically, these time limits are set to a value of less than one minute. A time-limit of one minute was established by operational users involved with the WSOA flight demonstration [34]. Thus, this metric will be key to determining *n*, the maximum number of tactical fighter nodes that can be adequately supported by the command and control node. In addition, this metric provides further context for the discussion of this issue in detail.

Another primary metric of concern is the compression level of image tiles that are transmitted across the Link-16 network via the WSOA architecture. The compression level of image tiles is important from a user perspective. If the WSOA architecture

cannot consistently deliver a majority of the image tiles at high resolution, i.e. a low image compression level, then the received imagery is not likely to be useful to the pilot, weapon systems officer or other operator on the aircraft [34].

3.7 Parameters

The parameters for the SUT are divided into two categories: system and workload. The system parameters are those that define the underlying system model and stay constant between simulation runs. As such, the system parameters are derived from technical specifications of the hardware and software that are components of the WSOA architecture. The workload parameters are those characteristics that affect the behavior of the workload. In this case, the workload parameters for the WSOA architecture are based on averages derived from actual workloads executed during live flight tests.

3.7.1 System

The WSOA architecture encompasses a number of system parameters, as depicted in Figure 3-1. The primary system parameters are the VTF imagery data, the JTIDS terminals, the Link-16 interface software, levels of compression utilized for VTF imagery tiles, and the scheduling algorithm used for providing timely service to multiple tactical nodes [34].

The imagery data being transmitted as part of the WSOA program consists primarily of images that are 512 x 512 pixels in size, and stored at 24 bits/pixel [34]. This results in an overall image size of 6,291,456 bits, and uncompressed image tiles of size 393,216 bits. VTF images are divided into 16 tiles.

The JTIDS terminals and Link-16 host interface software are system parameters

because their technical specification limits the performance of the WSOA architecture. Link-16 is a TDMA-based communication system. The basic unit of time in a Link-16 TDMA architecture is the epoch, which is defined to be 12.8 minutes [1]. Each time slot in a Link-16 TDMA ring is approximately 7.8125 ms [1]. A Link-16 TDMA ring is split up into three sets of timeslots: A, B and C [1]. Based upon the experimental design of the WSOA program, only one set of time slots is used. Using only one set of time slots provides 512 time slots per frame, with a frame length of 12s [1].

Another system parameter related to the QoS management framework is the levels of compression used to compress the VTF image tiles that are being transmitted across the Link-16 network. The compression levels used in this thesis, which are exactly the same compression levels used in the WSOA architecture are: 50:1, 75:1 and 100:1 [34]. Based on the image size described previously, these compression levels translate into image tile sizes that require approximately 7864 bits, 5243 bits, and 3932 bits, respectively. As such, these image tile sizes require 11, 8 and 6 Link-16 time-slots, respectively.

The scheduling algorithm used to service the imagery requests is also a system parameter. The scheduling algorithm used for this purpose is round-robin scheduling. This same scheduling algorithm will be used as the workload on the system is varied. Round-robin was chosen due to simplicity of implementation, and the lack of a defined scheduling algorithm within the WSOA architecture for supporting multiple tactical nodes. Other scheduling algorithms should be investigated as future research in this area.

3.7.2 Workload

The most significant workload parameter is the time associated with the

processing of image tiles. Image tile processing times are normally distributed based upon data from actual tests conducted on aircraft running the WSOA software architecture [34]. Image tile processing is divided into four separate parameters: tile queuing, tile decompression, QuO contract evaluation and QuO delegate execution. The timing parameters associated with QuO are the primary workload parameters being introduced to model the WSOA middleware architecture. Therefore, a sum of the parameters at a specific instance during the simulation represents an accurate model of the time required by the WSOA architecture to process a given image tile. Please refer to Table 3-1 for the specific averages and standard deviations associated with each of the timing parameters.

Table 3-1 System and Workload Parameters

System	Image Size	512 x 512 pixels, 24 bits/pixel
	Link-16 TDMA Epoch	12.8 minutes
	Link-16 TDMA Slot Length	7.8125 ms
	Link-16 TDMA Frame Length	12 s
	Imagery Compression Levels	50:1, 75:1, 100:1
	Scheduling Algorithm	Round-Robin
Workload	Tile Queuing	μ = 550.087 ms σ = 67087.693 ms
	Tile Decompression	μ = 17.344 ms σ = 6.324 ms
	QuO Contract Evaluation	μ = 78.203 ms σ = 3197.117 ms
	QuO Delegate Execution	μ = 124.844 ms σ = 6083.308 ms

3.8 Factors

There are two workload factors under consideration, number of tactical nodes and image deadline. The number of tactical nodes introduced into the system defines the workload of the system, since the command and control node is responsible for sending imagery data to all the tactical nodes requesting such data via the Link-16 communications network. The number of tactical nodes affects the number of receivers of imagery data, and the number of senders of QoS responses. This has a direct impact on the number of Link-16 TDMA slots that can be dedicated to a given tactical node, and thus the total number of tactical nodes that can be supported by the WSOA architecture. The number of tactical nodes that were introduced into the system ranged from 1 to 16. The case of a single tactical node is used to validate and verify the behavior of the model. The number of tactical nodes is then expanded in an exponential fashion, i.e., 2, 4, 8 and 16.

The deadline for downloading a complete image affects the calculations used by the WSOA architecture to determine whether the download of a given image tile is early, on-time or late. If the download of a given image tile is early or late, then appropriate transitions in the tactical nodes QoS state will occur, and those transitions will be communicated back to the C2 node. In turn, the C2 node will begin transmitting imagery to that tactical node at a different compression level. The overall image download time is varied between 38 – 54 seconds to control the workload on the system at a finer level of granularity.

Table 3-2 Workload Factors

Workload	Number of Tactical Nodes	1, 2, 4, 8, 16
	Image Deadline	38s, 42s, 46s, 50s, 54s

3.9 Evaluation Technique

The WSOA architecture under investigation has not implemented a scheduling algorithm, supporting the transmission of imagery data to multiple tactical nodes, to validate the results of the simulation against. The current research effort is being used to assess a "what-if" scenario, specifically to determine the maximum number of tactical nodes that the WSOA architecture can support. As such, the type of evaluation is simulation. The correctness of the modeled WSOA architecture is validated based upon the single tactical node case, since the WSOA architecture currently supports a single tactical node.

3.10 Experimental Design

The experiment uses the Link-16 TDMA communications model designed and implemented by Rockwell-Collins [7], which specifies all of the system parameters listed previously. This model also defines the workload based upon the bandwidth provided to a given node to receive and transmit data via the TDMA structure. Bandwidth is allocated to individual nodes via a slot map [1], which lays out recurrence rate numbers and indices create blocks of bandwidth.

In writing the code necessary to implement a functioning version of the WSOA architecture in OPNET®, all documentation relevant to the WSOA architecture is used to ensure the accuracy of the model. In addition, engineers from the Boeing Company, the

prime contractor responsible for implementing the WSOA architecture, were consulted when questions of implementation detail arose. The results are compared to existing data and test results measured from the WSOA architecture executing on actual aircraft. Any simplifications introduced to make the modeling more efficient or remove unnecessary functionality is documented.

After correctly implementing a functional version of WSOA architecture on top of the modeled Link-16 TDMA system, the experimental phase begins. Comparisons are based on a 90% confidence interval. Based on the stated factors, a full factorial experiment would require the number of experiments shown in Table 3-3.

Table 3-3 Experimental Design Determination

Image Download Time	Number of Tactical Nodes	Runs for CI	Total Experiments
38s	5 (1, 2, 4, 8, 16)	5	25
42s	5 (1, 2, 4, 8, 16)	5	25
46s	5 (1, 2, 4, 8, 16)	5	25
50s	5 (1, 2, 4, 8, 16)	5	25

3.11 Implementation Details

Implementing a complex software architecture, such as WSOA, requires that some assumptions be made and the parts of the architecture that are not implemented be documented and explained. The functionality associated with the Real-Time Adaptive Resource Manager (RTARM) and the dynamic scheduler was not implemented specifically in the model. The behavior of both RTARM and the dynamic scheduler are implicitly modeled through the image tile processing times. Since these image tile

processing times are based upon existing data from execution of the WSOA architecture, it is assumed that they model the behavior of RTARM and the dynamic scheduler.

3.11.1 Link-16 Communications Network

The wireless communication network shown in Figure 3-2 is similar to the communications network used during the WSOA ground and flight tests. The primary difference in the two communications networks is the number of tactical nodes, i.e., F-15s. In WSOA ground and flights tests, there is only one tactical node. In Figure 3-2 there are 16 tactical nodes, which are presented for the purposes of illustration. Other configurations are also similar with the primary difference being the number of tactical nodes.

Link-16 is a broadcast-type protocol so each node in the network can communicate with any other node that is within line-of-sight distance. All required system parameters are defined, to include the length of a timeslot, total number of timeslots, frame-size, number of timeslots in a given frame, etc., as discussed previously. Bandwidth is allocated to individual nodes through the use of a slot-map, which divides the bandwidth of the TDMA structure into usable blocks. The division of the bandwidth is accomplished via the use of Rate Recurrence Numbers (RRNs) and indices [1]. The RRNs divide a given frame of timeslots into blocks of timeslots, where each block of timeslots contains 2^{n-6} timeslots, with n being the RRN. The indices are used to address a given block of timeslots. For example, the ordered list of indices of timeslot blocks for RRN 12 is the following: 0, 4, 2, 6, 1, 5, 3, and 7 [1].

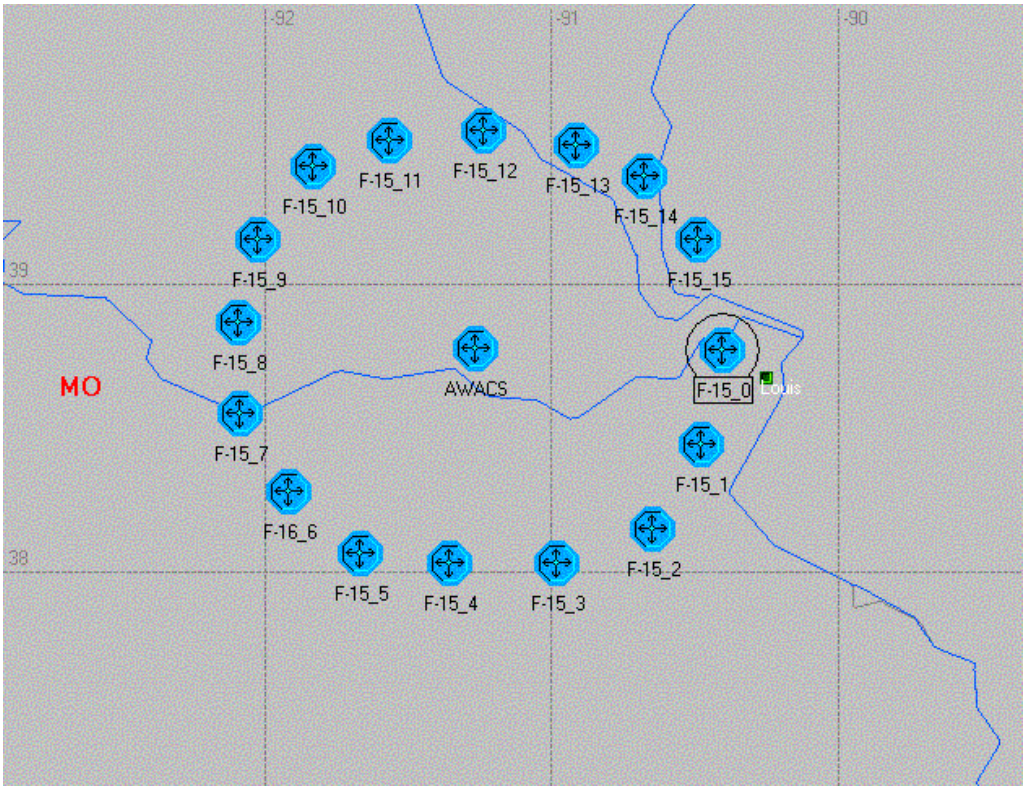

Figure 3-2. Example Link-16 Network with 16 Tactical Nodes

3.11.2 WSOA Object Request Broker (ORB) Packet

The WSOA ORB Packet message is a data packet used to simulate the transmission of imagery data to all relevant tactical nodes. The WSOA ORB Packet message contains fields for a source address, destination address, image tile number, image tile fragment number, response flag, tactical node QoS status, compression level associated with the simulated imagery data, and a time stamp.

The source and destination address fields are used by a node to determine if a given packet is addressed to that node. Since the communications network is limited to a single subnet that contains the C2 and all tactical nodes, no routing algorithm is required. The C2 node transmits simulated imagery data, and the tactical nodes transmit responses

based on this simulated imagery data. The image tile number and image tile fragment number are used to for the purposes of keeping track of the number of image tiles being sent to a given tactical node. Due to the Link-16 TDMA structure, a given image tile must be fragmented for transport across the network.

The response flag is used to determine if a given packet is a simulated imagery data packet transmitted from the C2 node, or a response packet transmitted from one of the tactical nodes. If the packet is a response from one of the tactical nodes, then the QoS status field contains information related to the current QoS status of that tactical node. Otherwise, the compression level field contains information related to the compression level of the current simulated imagery data being transmitted. The time stamp is used by a given tactical node as a part of its QoS early and late deadline calculations.

3.11.3 QoS Deadline Calculations and Adaptation

The QoS deadline calculations performed by the tactical nodes are used to determine the approximate number of image tiles that the tactical node should have received either ahead or behind schedule. Adaptation in the WSOA architecture, regarding the level of compression that imagery data is transmitted at, is controlled principally by these calculations [34]. The following formulas are those used in the WSOA architecture and implemented in the WSOA architecture model:

Early Deadline: (3.1)

Number of image tiles = (0.2 * Total image tiles) +
((Total image tiles/Maximum image download time) *
(Current total image download time))

Late Deadline: (3.2)

Number of image tiles = - (0.2 * Total image tiles) +
((Total image tiles/Maximum image download time) *
(Current total image download time))

If the number of the current image tile received by the tactical node is greater than the value calculated for the early deadline, then the QoS status is early. If the number of the current image tile received by the tactical node is less than the value calculated for the late deadline, then the QoS status is late. If the number of the current image tile received by the tactical node is greater than the value calculated for the late deadline, but less than the value calculated for the early deadline, then the QoS status is determined to be on-time [34]. Figure 3-3 illustrates the boundaries created by the early and late deadlines. In the figure, the lines labeled Image A and B represent two hypothetical images being downloaded via the WSOA architecture. I represents the percentage of the image which has been downloaded and processed by the tactical node. X and Z represent initial offsets, in terms of the percentage of a given image already downloaded and processed. These offsets demonstrate the convergence of the execution of the WSOA architecture to On-Time QoS region, and associated Y offset.

The above calculations are performed each time a complete image tile is received by a tactical node. This differs somewhat from the actual WSOA architecture, where the calculations occur on a much more frequent basis, due to the scheduling of tasks by the on-board computer in the WSOA architecture. Once the updated QoS status is determined by the tactical node, it is transmitted to the C2 node so that future imagery data can be transmitted at a level of compression appropriate for the tactical node. This feedback mechanism is the central adaptation mechanism in the WSOA architecture. The

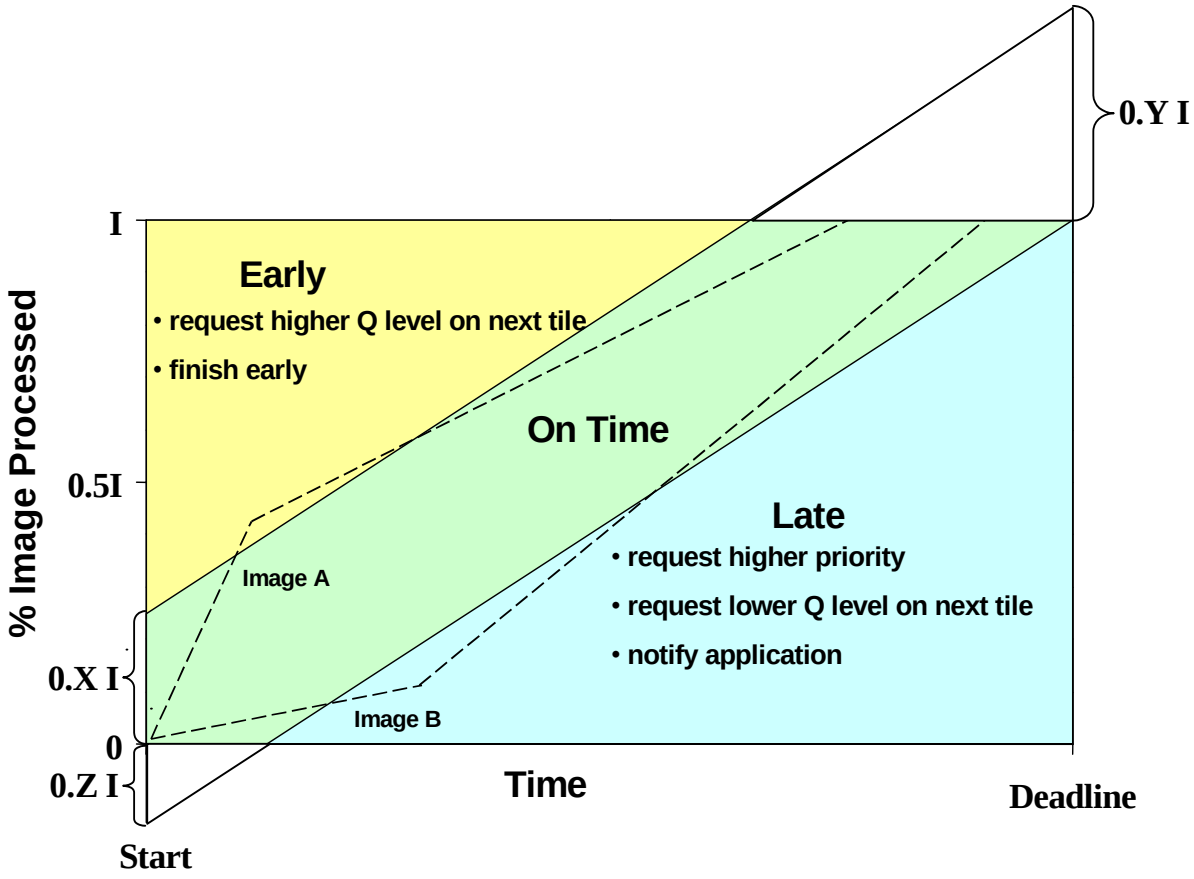

Figure 3-3. Early, On-Time and Late QoS Boundaries

Table 3-4 WSOA QoS Adaptation Model

Updated QoS Status	Current Compression Level	New Compression Level
Early	50:1	50:1
	75:1	50:1
	100:1	50:1
On-Time	50:1	50:1
	75:1	75:1
	100:1	100:1
Late	50:1	100:1
	75:1	100:1
	100:1	100:1

adaptation that is modeled, and occurs in the actual WSOA architecture, is depicted in Table 3-4.

3.12 Model Verification and Validation

Model verification was accomplished using a systematic approach. Simulation code was compiled for the target system. Problems with syntax and illegal statements were identified by the simulation environment and corrected. Once the models compiled correctly, the debugging cycle began.

The process of debugging began by implementing the capability to pass a WSOA ORB Packet message between the C2 node and a single tactical node. After designing and implementing the capability to send one WSOA ORB Packet message, the model was extended so that a single tactical node could send a response packet back to the C2 node. Once these first two steps were accomplished, then a basic feedback mechanism, very similar to the exact mechanism used in the WSOA architecture, was implemented and could be extended further. This is a brief overview of the major implementation milestones, but for the purposes of debugging, all of the following information was traced to verify that:

1. The C2 node transmitted the correct number of fragments for an image tile, at a given compression level. For image tiles compressed at 50:1, 75:1 and 100:1, the correct number of image tile fragments was 11, 8 and 6, respectively.
2. The tactical node performed the QoS Early and Late deadline calculations correctly and resulted in the tactical node transmitting a response packet that correctly reported the updated QoS status of the tactical node. For tactical nodes that updated their status to Early, On-Time or Late, the correct value associated with each status was 0, 1, and 2, respectively.

3. The C2 node maintained an accurate record, via an array, of the image tile numbers, image tile fragment numbers, compression levels, and QoS status for each tactical node.

4. The C2 node delivered the correct number of image tiles for a given tactical node. While this value could have been modified for the purposes of finer adaptation granularity, the correct number of image tiles was kept constant at 16.

5. The C2 node correctly performed the round-robin scheduling for all sets of tactical nodes, to include 1, 2, and 4 tactical nodes.

Model validation was accomplished using results and test data obtained from the Air Force Research Laboratory and Boeing, concerning actual ground and flight tests conducted on the WSOA architecture. Three elements of the model must be validated [12]:

1. Assumptions,
2. Input parameter values and distributions, and
3. Output values.

Since a working implementation of the WSOA architecture existed, then no major assumptions had to be made concerning the model of the WSOA architecture. All implementation details and questions could be answered either through existing documentation or consultation with engineers at the Air Force Research Laboratory or the Boeing Company.

Underlying network model validation was accomplished by sending WSOA ORB Packet messages back and forth between a single C2 node and a multiple tactical nodes. Source and destination addresses were assigned sequentially and packets were sent and received by all tactical nodes.

Input parameters for the image tile processing times were chosen to closely match the parameters used in the WSOA ground and flight tests [34]. The choice of distributions for each image tile processing parameter was developed from statistical analysis, which in-turn was based on actual test data and results.

Output values used to validate the model consisted primarily of the compression levels of the simulated imagery data for a single tactical node. Validation tests were run with overall image deadlines of 38, 42, 46, and 50 seconds. The values of the compression levels for a single tactical node were compared to the values that were recorded during the WSOA ground and flight tests.

In general, the simulation results matched the results from the WSOA ground and flight tests. Slight variations did occur, but can be attributed to the granularity of time that the Early/Late deadline calculations were performed at. As explained previously, the calculations in the simulation were performed on a periodic basis, while the calculations that occurred in the actual WSOA ground and flight tests were performed on a periodic basis with a much shorter period.

3.13 Summary

This chapter presented the methodology for the experimental stage of this thesis. Additional background information regarding the goals and hypothesis, system

boundaries and system services was presented. Performance metrics, parameters, factors, experimental design, implementation details, and validation and verification of the model were all presented and described in detail.

4. Analysis

4.1 Introduction

This chapter presents simulation results and analysis. Before explaining the simulation results, a brief overview of the statistical methods used is presented. Following this overview, the results from the image tile performance measurements are presented. All three metrics will be presented in the context of 1, 2, 4, 8 and 16 tactical nodes. The conclusion of this chapter discusses the original research goal, to determine a value for *n*, and the relationship between this value and the allocation of TDMA bandwidth.

4.2 Statistical Overview

This section explains the methods used to determine results and provides a brief overview of how statistical values are generated and applied. Pilot studies and preliminary simulations were run to determine the transient period of the simulation. In Figure 4-1, the transient period was over within the first 300 seconds of simulation time.

4.2.1 Simulation Statistics

Simulation sets are divided into five groups, based on overall image deadline. Groups are subdivided into five distinct loading levels, based on number of tactical fighter nodes. Each group is executed five times, with different random seeds, to achieve the desired confidence interval width, and yielding 125 total experiments.

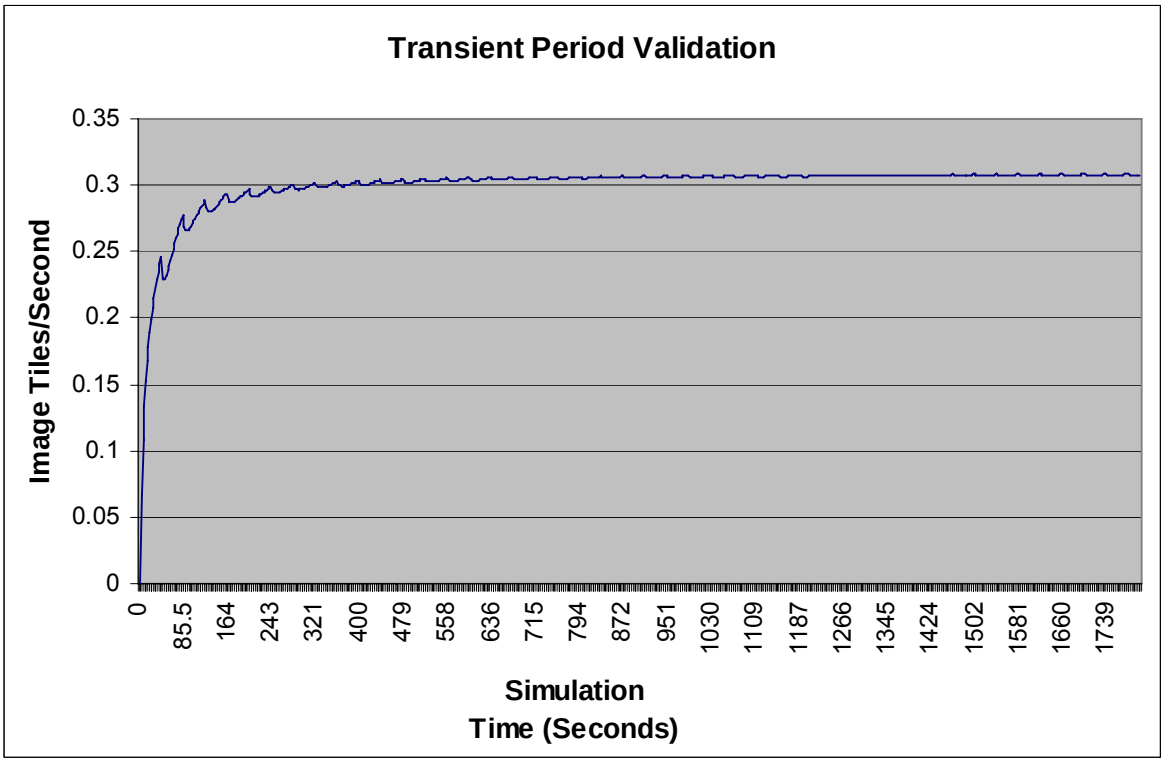

Figure 4-1. Transient Period Validation – Image Tiles Per Second

4.2.2 Confidence Intervals

The confidence level chosen for this research is 90%. A 90% confidence level indicates that for any mean, there is a 90% probability that the actual mean lies inside the interval [12]. The following equation defines the confidence interval

$$\left(\bar{x} - z_{1-\frac{\alpha}{2}} \, s/\sqrt{n}, \bar{x} + z_{1-\frac{\alpha}{2}} \, s/\sqrt{n} \right) \qquad 4.1$$

where $\bar{x}$ is the sample mean, $z_{1-\frac{\alpha}{2}}$ is the $\left(1-\frac{\alpha}{2}\right)$ quantile of a unit normal variate, σ is the variance, s is the standard deviation, and n is the number of samples. If the confidence

interval computed for one mean contains the second mean, then the two items being compared can be considered statistically equivalent. If a given confidence interval does not contain the mean, then the items being compared may be considered statistically different at the given level of confidence.

4.2.3 Coefficient of Variation

The Coefficient of Variation (C.O.V.) [12], is the ratio of standard deviation to sample mean, which is defined by the following equation:

$$C.O.V. = \frac{s}{\bar{x}} \qquad 4.2$$

A C.O.V. of less than 10% is used as a stopping criterion for simulations.

4.2.4 Analysis of Variance

ANalysis Of VAriance (ANOVA) is used to determine interactions between the primary effects, secondary effects, and tertiary effects [12]. ANOVA is a method to calculate the variance attributable to each experimental factor, and assign each experimental factor a percentage of the total variation. Factors can be classified by the resulting experimental effects that are observed. A single factor is the source of primary effects, interactions between two factors contribute to secondary effects, and as such, interactions between three factors result in the tertiary effects. The sum of the squares for the determined effect is divided by the total sum of squares for all effects. The final step in the analysis is to perform an F-test to determine the significance of the allocation at the given significance level. The ANOVA analysis is only valid if the assumptions below are satisfied:

1. Residuals versus predicted responses should show no trend when plotted on a scatter plot,

2. Normal quantile-quantile plot should show a straight line of data points with little (or no) deviation.

The method of calculating ANOVA tables is presented below for a two factor experiment [12]. Equation 4.3 is the total sum of squares for both factors. Equations 4.4 and 4.5 show the primary sum of square effects for factors A and B. Equation 4.6 shows the combined sum of squares effect for factor AB.

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{b} y_{ij}^2 - \frac{y^2}{ab} \qquad 4.3$$

$$SS_A = \frac{1}{b}\sum_{i=1}^{a} y_{i\ldots}^2 - \frac{y^2}{ab} \qquad 4.4$$

$$SS_B = \frac{1}{a}\sum_{j=1}^{b} y_{j\ldots}^2 - \frac{y^2}{ab} \qquad 4.5$$

$$SS_{AB} = \sum_{i=1}^{a}\sum_{j=1}^{b} y_{ij\ldots}^2 - \frac{y^2}{ab} - SS_A - SS_B \qquad 4.6$$

4.2.5 Random Methods

Stochastic methods were used to generate the image tile service times to include tile queuing, tile decompression, QuO contract evaluation and QuO delegate execution. These image tile processing time parameters were modeled as being normally distributed based upon data from actual tests conducted on aircraft running the WSOA software architecture [34]. By seeding the simulation runs differently for the five separate trials, the values generated for each time parameter are different for each simulation iteration, but still follow the distributions identified as characterizing the existing test data.

4.3 WSOA Image Deadline Scenarios

There are five scenarios that simulate the behavior of the WSOA architecture, in terms of deadlines for downloading a complete image. The following deadlines are used for image downloads (in seconds): 38, 42, 46, 50, and 54. These values are chosen as the image download deadlines because these values are the same as those used during the ground and flight testing conducted on the WSOA architecture.

4.3.1 Image Tiles Per Second Analysis

As discussed previously, the Image Tiles Per Second analysis is replicated for 1, 2, 4, 8 and 16 tactical nodes.

Figure 4-2 shows the results for the Image Tiles Per Second metric for each of the respective image deadline experiments. As demonstrated by the experimental results, the number of tactical nodes does not impact the overall performance of the WSOA architecture in the cases of 1, 2 and 4 tactical nodes. There is a slight reduction in throughput for the 1, 2, and 4 node experiments across the various image deadlines. As the deadline is extended from 38 seconds to 42 seconds and so on, the overall throughput for the system is reduced because a given image is allowed more time for downloading.

Initially, the performance of the WSOA architecture does seem to be impacted significantly by the number of tactical nodes in the 8 and 16 node cases. This can be attributed to the amount of TDMA bandwidth allocated in these cases. In the 2 and 4 node cases, the amount of TDMA bandwidth allocated to each tactical node is equal to the bandwidth allocated to a single tactical node. In the 8 and 16 node cases, the amount of TDMA bandwidth allocated to each tactical node is not equal because there is not enough bandwidth for such an allocation. Therefore, the tactical nodes in these cases are

forced to share bandwidth. As a consequence, the scheduling algorithm used to service the imagery requests affects the performance of the WSOA architecture. As will be discussed later, TDMA bandwidth should have been considered as separate factor to be studied independent of the number of tactical nodes. A different scheduling algorithm might be able to provide some improvement in the performance of the WSOA architecture. But, this type of modification is unlikely to improve the performance to the level observed in the 1, 2, and 4 node cases. Thus the performance of the WSOA architecture does scale well for this metric, based on the assumption that each tactical node is allocated sufficient bandwidth.

All of the results presented, across each of the image deadlines, are within the 90% confidence interval, and thus can be considered statistically identical. This behavior is confirmed by ANOVA analysis (c.f., Appendix A) which finds that the overall image deadline accounts for 0.67% of the variance for each experiment. The number of tactical nodes accounts for 98.72% of the variance for each experiment. The maximum average value for the image tiles per second metric is approximately 0.17. This metric is derived by measuring the number of image tiles that are received during a given period that a single image is downloaded.

These results, given the respective image deadlines, are expected. The throughput of the WSOA architecture, as measured by image tiles per second, is considered satisfactory for the 1, 2, and 4 node cases. The throughput of the WSOA architecture is not satisfactory in the cases of 8 and 16 nodes. Further simulation and analysis is required to determine for certain that the results obtained in the 8 and 16 nodes cases can be attributed directly to the allocation of TDMA bandwidth. Once accomplished, then a

definitive statement as to overall scalability of the WSOA architecture can be made with regard to this performance metric.

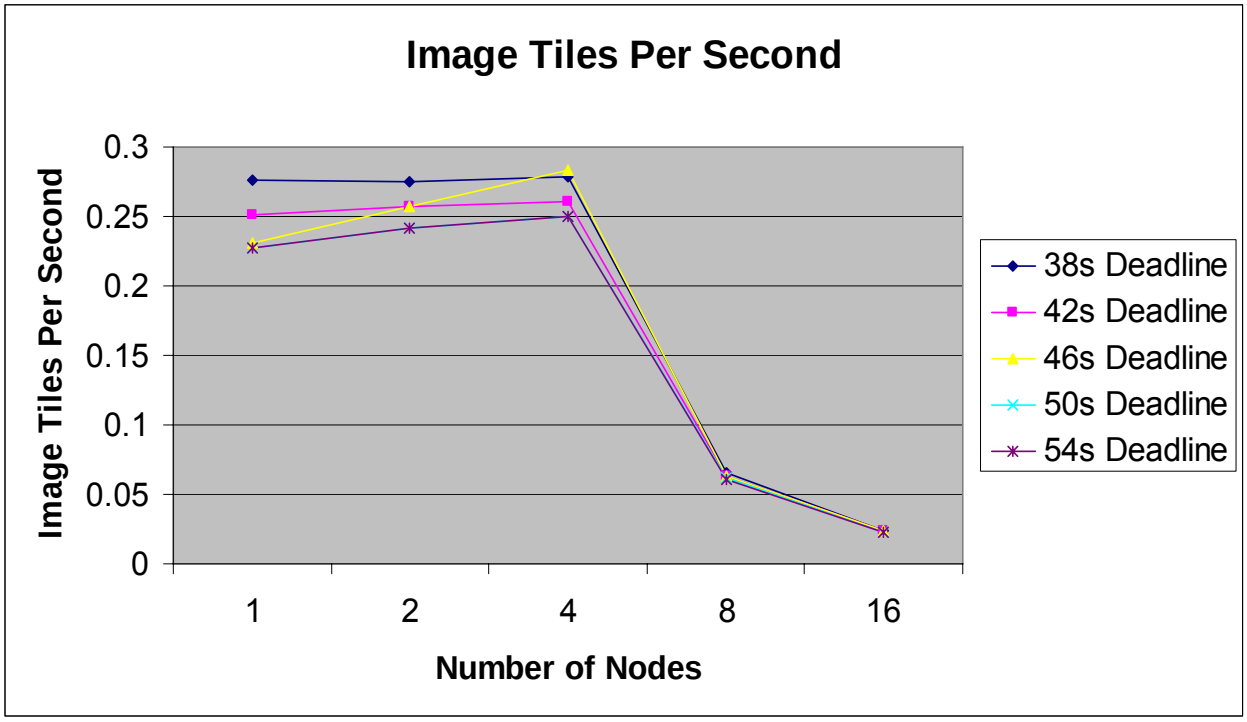

Figure 4-2. Image Tiles Per Second Results

4.3.2 Compression Level Analysis

As discussed previously, the Compression Level analysis is replicated for 1, 2, 4, 8 and 16 tactical nodes. The Compression Level metric is an average of the compression levels measured for the image tiles being transmitted. The compression levels used by the WSOA architecture are discrete, i.e. 50:1, 75:1 and 100:1, but an average of the recorded values provides relative insight into overall image resolution. A compression level average closer to 50 would indicate that the overall image resolution is nearly maximum,

while conversely, a compression level average closer to 100 would indicate that the overall image resolution is nearly minimum.

Figure 4-3 shows the results for the Compression Level metric for each of the respective image deadline experiments. As demonstrated by the results, the number of tactical nodes does not impact the adaptation strategy of the WSOA architecture in the cases of 1, 2 and 4 tactical nodes. There is a reduction in the metric results for the 1, 2, and 4 node experiments across the various image deadlines. As the deadline is extended from 38 seconds to 42 seconds and so on, the average compression level for individual image tiles is reduced because the image is allowed more time for downloading. Thus, the WSOA architecture has more flexibility in regards to selecting the compression level for a given image tile.

Initially, the adaptation strategy of the WSOA architecture does seem to be impacted significantly by the number of tactical nodes in the 8 and 16 node cases. Once again, this can be attributed to the amount of TDMA bandwidth allocated in these cases. The allocation of TDMA bandwidth has a significant effect on the compression level for individual image tiles for the same reasons provided in the analysis of the Image Tiles Per Second metric. Again, TDMA bandwidth should have been considered as a separate factor to be studied independent of the number of tactical nodes. Thus the performance of the WSOA architecture does scale well for this metric, based on the assumption that each tactical node is allocated sufficient bandwidth.

All of the results presented, across each of the image deadlines, are within the 90% confidence interval, and thus can be considered statistically identical. This behavior is confirmed by ANOVA analysis which finds that the image deadline accounts for

14.09% of the variance for each experiment. The number of tactical nodes accounts for 76.80% of the variance for each experiment. The maximum average value for the compression level metric is approximately 75.36.

These results, given the respective image deadlines, are expected. The average compression level of the WSOA architecture is considered satisfactory for the 1, 2, and 4 node cases. The average compression level of the WSOA architecture is not satisfactory in the cases of 8 and 16 nodes. Further simulation and analysis is required to determine for certain that the results obtained in the 8 and 16 nodes cases can be attributed directly to the allocation of TDMA bandwidth. Once accomplished, then a definitive statement as to overall scalability of the WSOA architecture can be made with regard to this performance metric.

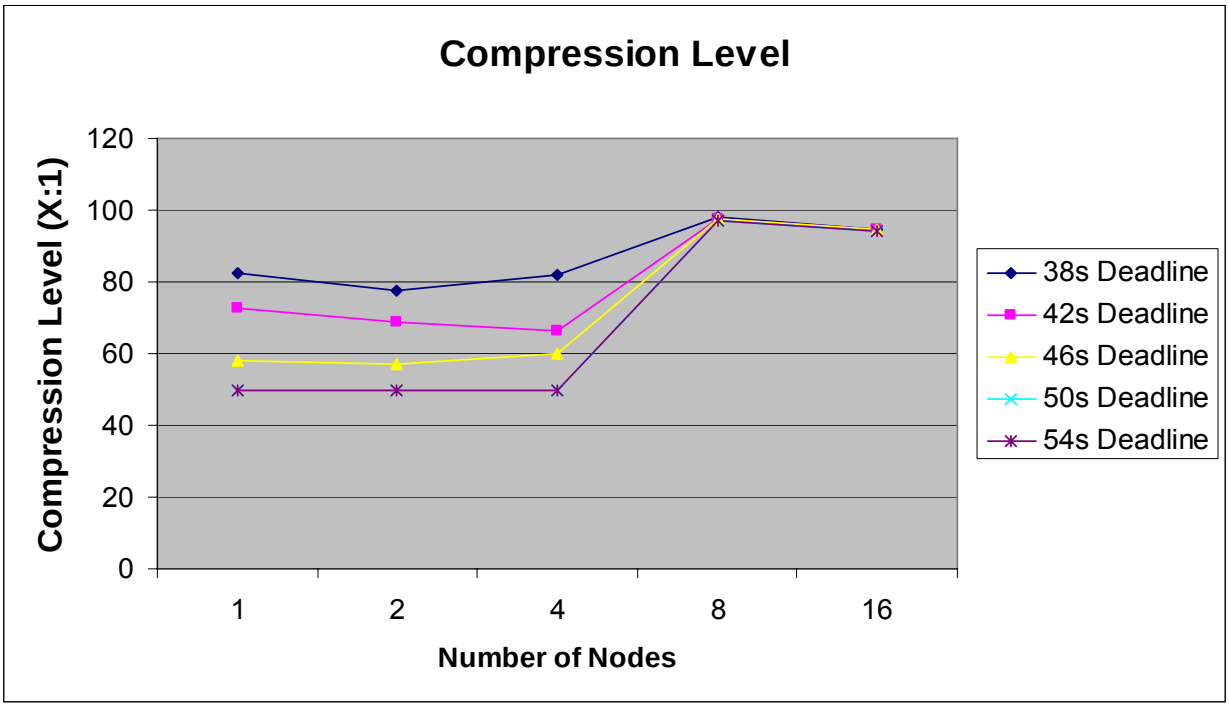

Figure 4-3. Compression Level Results

4.3.3 Image Download Time Analysis

The Image Download Time analysis is replicated for 1, 2, 4, 8 and 16 tactical nodes. The Image Download Time metric measures the time required by a given tactical fighter node to download a single complete image.

Figure 4-4 shows results for the Image Download Time metric for each of the respective image deadline experiments. As demonstrated by the results in the cases of 1, 2 and 4 tactical nodes, the number of nodes does not affect the overall download time for a given image transmitted by the WSOA architecture.

There is a slight increase in the metric results for the 1, 2, and 4 node experiments across the various image deadlines. As the deadline is extended from 38 seconds to 42 seconds and so on, the overall image download time increases proportional to the increase in the image deadline.

Initially, the overall download time for a given image transmitted by the WSOA architecture does seem to be impacted significantly by the number of tactical nodes in the 8 and 16 node cases. Once again, this can be attributed to the amount of TDMA bandwidth allocated in these cases. The allocation of TDMA bandwidth has a significant effect on the image download time for the same reasons provided in the analysis of the Image Tiles Per Second and Compression Level metrics. Again, TDMA bandwidth should have been considered as a separate factor to be studied independent of the number of tactical nodes. Thus the performance of the WSOA architecture does scale well for this metric, based on the assumption that each tactical node is allocated sufficient bandwidth.

All of the previous results are within the 90% confidence interval, and thus can be considered statistically identical. This behavior is confirmed by ANOVA analysis which finds that the overall image deadline accounts for 0.09% of the variance for each

experiment. The number of tactical nodes accounts for 99.86% of the variance for each experiment. The maximum average value for the image download time metric is approximately 181.65s.

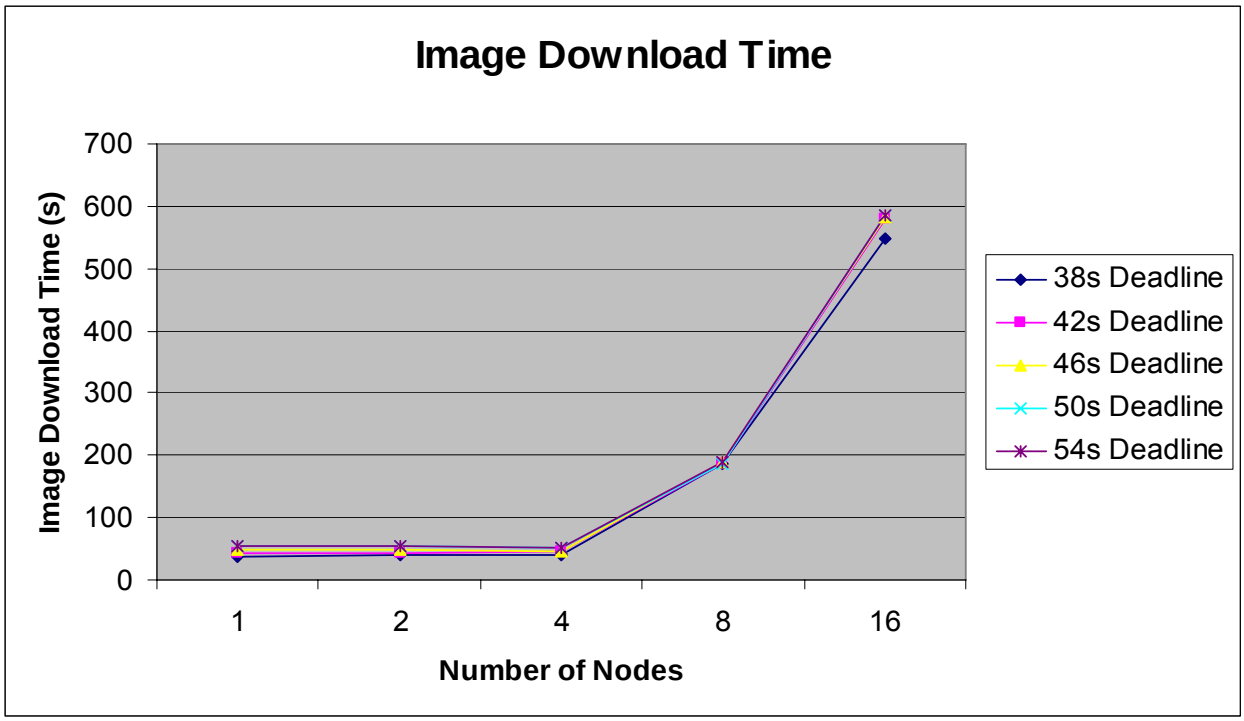

Figure 4-4. Image Download Time Results

These results, given the respective image deadlines, are expected. The overall download time for a given image transmitted by the WSOA architecture is considered satisfactory for the 1, 2, and 4 node cases. The overall download time for a given image transmitted by the WSOA architecture is not satisfactory in the cases of 8 and 16 nodes. Further simulation and analysis is required to determine for certain that the results obtained in the 8 and 16 nodes cases can be attributed directly to the allocation of TDMA bandwidth. Once accomplished, then a definitive statement as to overall scalability of the WSOA architecture can be made with regard to this performance metric.

4.4 Conclusion

The WSOA architecture provides a scalable framework for the transmission of real-time imagery and other complex data products from command and control aircraft, such as AWACS or JSTARS, to tactical aircraft, such as F-15s or F-16s. As demonstrated by the results, the WSOA architecture scales well with the increase in the number of tactical nodes that are supported by the architecture, in the 1, 2 and 4 node cases. In the cases of 8 and 16 tactical nodes, the performance of the WSOA architecture initially seemed to be impacted significantly by the number of nodes. While this is statistically true in regards to all three of the metrics collected in the context of the analysis that was performed, the actual explanation for the impact on performance is related to the allocation of TDMA bandwidth.

As discussed previously, further simulation and analysis is required to determine for certain that the results obtained in the 8 and 16 nodes cases can be attributed directly to the allocation of TDMA bandwidth. This will require additional research and work to modify the existing simulation model to support a "low-bandwidth" TDMA allocation in the 1, 2, and 4 node cases. This simulation of this "low-bandwidth" TDMA allocation will provide the additional data required to perform a complete analysis that can clearly demonstrate that bandwidth is the factor that has the greatest effect on the scalability of the WSOA architecture. At this point, only in the 1, 2, and 4 node cases can one conclude that the WSOA architecture still scales well, regardless of the number of tactical nodes.

5. Conclusions

5.1 Restatement of Research Goal

The principal goal of this research effort is modeling and simulating the WSOA architecture, to determine its scalability as a networking protocol for DRE systems. The current WSOA architecture supports a single command and control aircraft and a single tactical fighter node. For the purposes of demonstrating the application of new technology, this type of limited experimental setup is sufficient. Since this technology will eventually transition to existing military systems, questions concerning the scalability of the WSOA architecture and underlying technology must be explored. Specifically, the goal of this study was to determine the maximum number of tactical fighter nodes that can be supported, at varying levels of QoS, by a given command and control node.

5.2 Research Contribution

This research is the first to implement and analyze the WSOA middleware architecture in a network simulation environment. This work also introduces a simple round-robin scheduling algorithm to transmit image tiles to multiple tactical nodes. While round-robin scheduling is certainly not unique, this type of scheduling is the first to be implemented in the context of the WSOA architecture supporting multiple tactical nodes. In addition, this forms the foundation for future research involving other more pertinent scheduling algorithms, when such algorithms are eventually identified.

5.3 Conclusion

A scalable protocol is a critical component in any information infrastructure, especially in the case of an infrastructure that is attempting to disseminate information in real-time. As implemented here, the WSOA architecture provides this capability for up to 4 tactical nodes. This successful demonstration of the WSOA architecture is due in large part to the amount of Link-16 TDMA bandwidth that is allocated to each tactical node. In the 2 and 4 node cases, the amount of TDMA bandwidth is essentially equal to that which is allocated in the single node case. Thus, in experiments with increasing numbers of tactical nodes, the nodes are required to share the available Link-16 TDMA bandwidth. As demonstrated in the 8 and 16 node cases, this sharing of bandwidth has a significant impact on the performance of the WSOA architecture.

Given the explanation and justification above, one can conclude that the number of tactical nodes alone did not affect the performance of the WSOA architecture in any significant fashion. Thus, the WSOA architecture effectively adapted to changes in the deadline set for the overall download time for a single image, regardless of the number of tactical nodes.

5.4 Future Research

Many facets of the WSOA architecture lend themselves to areas for future research and improvement. The most obvious future research effort is to continue experimenting with the number of tactical nodes that the WSOA architecture supports in simulation, so as to determine a value for the parameter *n* that is further refined for different operational contexts. Based on the results of this effort and the context given, an estimated value for this parameter falls in the range between 4 and 8 tactical nodes.

5.4.1 Scheduling Algorithms

Once sufficient research has been completed in the area of applying dynamic TDMA bandwidth allocation strategies to military tactical datalinks such as Link-16, then the research completed and documented here on the WSOA architecture should be revisited. The admission of tactical nodes into the existing communications infrastructure will likely be the deciding factor in choosing a scheduling algorithm for ordering the transmission of image tiles by the C2 node. Experimentation in this area could be performed in the near future, but should be directed by the results of on-going research to add dynamic bandwidth allocation strategies to existing military tactical datalinks. Possible scheduling algorithms include priority-based schemes, real-time schemes (RMA, EDF, etc.) and just about any other applicable scheduling algorithm.

5.4.2 Military Tactical Datalinks

Given that the implementation of the WSOA architecture presented here is fairly modular, another interesting area of research would be to substitute models of other military tactical datalinks for the Link-16 model presented here. In all likelihood, the C2 node will be supporting tactical nodes that are acting as flight leads for particular strike packages or other arrangements of aircrafts. As such, it may be the job of a flight lead aircraft to disseminate real-time information transmitted across the WSOA architecture to other tactical nodes in the strike package. Thus, studying the performance of the WSOA architecture in the context of other military tactical datalinks should also be explored.

Appendix A. Data

Table A-1. WSOA Architecture Performance Metrics

Image Deadline	# of Tactical Nodes	Image Tiles Per Second		Compression Level		Image Download Time	
		μ	Σ	μ	σ	μ	Σ
38 Seconds	1	0.27604	0.00000	82.42686	0.07714	38.62302	0.01254
	2	0.27544	0.00000	77.70597	0.13251	40.02178	0.02434
	4	0.27877	0.00000	81.96039	0.01315	40.38903	0.00578
	8	0.06554	0.00000	97.98015	0.00003	185.46009	0.00008
	16	0.02379	0.00000	94.44008	0.04978	548.32441	3.70920
42 Seconds	1	0.25121	0.00000	72.44048	0.24871	43.24858	0.03392
	2	0.25658	0.00000	68.79808	0.19755	44.32527	0.04181
	4	0.26071	0.00000	66.42475	0.02364	45.07008	0.00679
	8	0.06334	0.00000	97.52994	0.00000	187.76057	0.00973
	16	0.02361	0.00000	94.66064	0.00959	580.35583	0.54118
46 Seconds	1	0.23120	0.00000	58.20557	0.03471	50.18691	0.00228
	2	0.25685	0.00001	56.96875	0.46322	49.76129	0.12759
	4	0.28294	0.00001	59.99566	0.36036	47.30541	0.12676
	8	0.06262	0.00000	97.37964	0.00034	188.44555	0.00978
	16	0.02334	0.00000	94.64290	0.00746	582.11194	0.51355
50 Seconds	1	0.22734	0.00000	50.00000	0.00000	54.75167	0.00166
	2	0.24160	0.00000	50.00000	0.00000	53.56711	0.00313
	4	0.25047	0.00000	50.00000	0.00000	52.90575	0.00081
	8	0.06148	0.00000	97.15617	0.00074	186.68725	0.01612
	16	0.02311	0.00000	94.24186	0.00108	586.28092	0.11814
54 Seconds	1	0.22734	0.00000	50.00000	0.00000	54.75167	0.00166
	2	0.24160	0.00000	50.00000	0.00000	53.56711	0.003134
	4	0.25047	0.00000	50.00000	0.00000	52.90575	0.00081
	8	0.06058	0.00000	96.88285	0.00037	188.12161	0.00003
	16	0.02297	0.00000	94.19334	0.00067	586.37747	0.92064

Note: An * after the percentage denotes the effect was significant based on the computed F-Test

Table A-2. ANOVA Analysis for 38, 42, 46, 50 and 54 Second Trials

		Image Tiles Per Second	Compression Level	Image Download Time
Main Effects	Overall Image Deadline	0.67% *	14.09% *	0.09% *
	# of Tactical Nodes	98.72% *	76.80% *	99.86% *
Unaccounted		0.61%	9.11%	0.05% *

Example A-2. Image Tiles Per Second ANOVA

Computation of Effects

		Image Deadline							
		38	42	46	50	54	Row Sum	Row Mean	Row Effect
# of Nodes	1	0.2760	0.2512	0.2312	0.2273	0.22734	1.2131	0.2426	0.0730
	2	0.2754	0.2565	0.2568	0.2416	0.2416	1.2720	0.2544	0.0848
	4	0.2787	0.2607	0.2829	0.2504	0.25047	1.3233	0.2646	0.0951
	8	0.0655	0.0633	0.0626	0.0614	0.06058	0.3135	0.0627	-0.1068
	16	0.0237	0.0236	0.0233	0.0231	0.02297	0.1168	0.0233	-0.1461
Column Sum		0.919	0.855	0.856	0.804	0.80296			
Column Mean		0.183	0.171	0.171	0.160	0.16059		0.1695	
Column Effect		0.014	0.001	0.001	-0.008	-0.0089			

Estimating Experimental Error

		Image Deadline						
		38	42	46	50	54		
# of Nodes	1	0.019	0.007	-0.013	-0.006	-0.0063		
	2	0.006	0.000	0.000	-0.004	-0.0038		SSE
	4	0.000	-0.005	0.016	-0.005	-0.0052		0.0016
	8	-0.011	0.000	-0.001	0.007	0.00683		
	16	-0.014	-0.001	-0.002	0.009	0.0086		

Allocation of Variation

SST = SSY-SS0
SSY = SS0 + SSA + SSB + SSE

SSY =	0.994154
SS0 =	0.718744
SSA =	0.001844
SSB =	0.271874
SST =	0.275410
SSE =	0.001690

Var.	% Deadline	0.669
	% # Nodes	98.71
	% Error	0.613
	Total	100

Analysis of Variance

MSA =	0.000461
MSB =	0.067968
MSE =	0.000105
FA =	4.363842
FB =	643.1368

Component	Sum of Sqrs.	% of Var.	Degrees	Mean Sqr	F-Comp	F-Table
y	0.994154					
ybar	0.718744					
y-ybar	0.275410	100	24			
Dead.	0.001844	0.669	4	0.000	4.363	2.33
# Nodes	0.271874	98.71	4	0.067	643.1	2.33
Errors	0.001690	0.613	16	0.000		

Appendix B. Availability of OPNET® Models and Source Code

OPNET® Models and source code are not included as part of this document. Interested parties should direct their inquiries to:

Dr. Richard Raines

AFIT/ENG

2950 Hobson Way, Bldg 642

Wright-Patterson AFB, OH 45433-7765

Bibliography

[1] Barto, Jon L, *Air Force JTIDS Network Library Design Guide*, revision 2, MTR-10872, Mitre: Bedford, Massachusetts, September 1991.

[2] Bittel Raymond, Caples Edward, Young C. David, Loso Frank, Soldier Phone: An Innovative Approach to Wireless Multimedia Applications, *Proceedings of IEEE MILCOM 1998*, October 1998.

[3] Box, D., *Essential COM*, Addison-Wesley, Reading, MA, 1997.

[4] Cardei I., Jha R., Cardei A. Pavan, Hierarchical Architecture for Real-Time Adaptive Resource Management. *Proceedings of the ACM/IFIP Middleware 2000 Conference*, Palisades, NY, April 2000.

[5] Corman Dr. David, Gossett Jeanna, Weapon System Open Architecture – Using Emerging Open System Architecture Standards to Enable Innovative Techniques for Time Critical Target Prosecution. *Proceedings of 2002 IEEE Digital Avionics Systems Conference*.

[6] DARPA, "The Quorum Program", http://www.darpa.mil/ito/research/quorum/index.html, 1999.

[7] *DNS-16 Final Report*. Rockwell-Collins. 2003

[8] *ECE 678: Integrated Telecommunications Networks*. http://www.ece.arizona.edu/~ece678/notes/Projects2003.ppt

[9] Gill Christopher, Loyall Joseph, Schantz Richard, and Schmidt Doug. Experiences Using Adaptive Middleware in Distributed Real-time Embedded Application Contexts: a Dependability Perspective. Workshop on Dependable Middleware-Based Systems, Part of Dependable Systems and Networks Conference (DSN 2002), June 26, 2002, Bethesda, Maryland.

[10] Horowitz, Eric J., *Modeling Object-Oriented Architectures With OPNET®*. http:///www.citeseer.nj.nec.com/527123.html.

[11] Hura Myron, et al. *Interoperability-A Continuing Challenge in Coalition Air Operations*. RAND Corporation. 2000.

[12] Jain, Raj. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons. 1991.

[13] Kiczales G., Towards a New Model of Abstraction in the Engineering of Software. *Proceedings of the Workshop on Reflection and Meta-Level Architectures (IMSA '92)*, 1992.

[14] Klein M. H., Ralya T., Pollak B., Obenza R., Harbour M.G., *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Norwell MA, Kluwer Academic Publishers, 1993.

[15] Levine David L., Gill D. Christopher, Schmidt Douglas C., Dynamic Scheduling Strategies for Avionics Mission Computing. *Proceedings of 1998 IEEE Digital Avionics Systems Conference*.

[16] Leydekkers P., Gay V., Franken L., A Computational and Engineering View on Open Distributed Real-Time Multimedia Exchange. *Proceedings of the Fifth International Workshop on Network and Operating System Support for Digital and Audio and Video (NOSSDAV '95)*, Boston USA, April 1995.

[17] Liu C., Layland J., Scheduling Algorithms for Mulitprogramming in a Hard Real-Time Environment. *Journal of the ACM*, Vol. 20, pg 46-61, January 1973.

[18] Lindholm T., Yellin F., *The Java Virtual Machine Specification*, Addison-Wesley, Reading, MA, 1997.

[19] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, 2.2 ed., Feb 1998.

[20] Object Management Group. "CORBAServices: Common Object Service Specification," OMG Technical Document formal/98-12-31.

[21] OPNET®, http://www.opnet.com

[22] Pond L.C., Li V.O.K. A Distributed Time-Slot Assignment Protocol for Mobile Multi-Hop Broadcast Packet Radio Networks, *Proceedings of IEEE MILCOM 1989*, Vol. 1, November 1989.

[23] Schantz Richard, Quality of Service, a survey article prepared for *Encyclopedia of Distributed Computing*, Kluwer Academic Publishers, Partha Dasgupta and Joseph Urban, editors. 1999.

[24] Schantz R.E. , Schmidt D. C., Middleware for Distributed Systems – Evolving the Common Structure for Network-centric Applications. Chapter in *The Encyclopedia of Software Engineering*. John Wiley & Sons. December 2001.

[25] Sharp David C. *Reducing Avionics Software Cost Through Component Based Product Line Development*. 1998 Software Technology Conference. April 1998.

[26] Schmidt D., Huston S., *C++ Network Programming: Resolving Complexity with ACE and Patterns*, Addison-Wesley, Reading, MA, 2001.

[27] Snell J., MacLeod K., *Programming Web Applications with SOAP*, O'Reilly, 2001.

[28] Stewart D. B., Khosla P. K., Real-Time Scheduling of Sensor-Based Control Systems, *Real-Time Programming* (W. Halang and K. Ramamritham, eds.), Tarrytown NY, Pergamon Press, 1992.

[29] Thai T. Lam H., *.NET Framework Essentials*, O'Reilly, 2001.

[30] Thomas, Anne. "Enterprise Java Bean Technology", http://java.sun.com/products/ejb/white_paper.html, Dec. 1998.

[31] Tuma Petr, Buble Adam. Overview of the CORBA Performance. http://citeseer.nj.nec.com/tuma02overview.html.

[32] Wang Nanbor, Schmidt Douglas C., Gokhale Aniruddha, Gill Christopher D., Balachandran Natarajan, Rodrigues Craig, Loyall Joseph, and Schantz Richard E. Total of Quality of Service Provisioning in Middleware and Applications. *The Journal of Microprocessors and Mircrosystems*, Elsevier, vol. 26, number 9-10, January 2003.

[33] Wollrath A., Riggs R., Waldo J. "A Distributed Object Model for the Java System," *USENIX Computing Systems*, 9(4), 1996.

[34] Weapon System Open Architecture (WSOA) Final Report. Boeing Company. 2002

[35] Young C. David, USAP: A Unifying Dynamic Distributed Multichannel TDMA Slot Assignment Protocol. *Proceedings of 1996 IEEE Digital Avionics Systems Conference*.

[36] Zinky John, Bakken David, Schantz Richard. Architectural Support for Quality of Service for CORBA Objects. Theory and Practice of Object Systems, April 1997.

Vita

Captain Jason T. Lawson graduated from Lyman Hall High School in Wallingford, Connecticut. He entered undergraduate studies at the University of Connecticut in Storrs, Connecticut where he graduated with a Bachelor of Science degree in Engineering in December 1998. He was commissioned through the Detachment 115 AFROTC at the University of Connecticut.

His first assignment was at Wright-Patterson AFB as a HW/SW Systems Research Engineer for the Information Directorate of the Air Force Research Laboratory. in March 1999. In August 2002, he entered the Graduate School of Engineering and Management, Air Force Institute of Technology. Upon graduation, he will be assigned to the Air Intelligence Agency, Lackland AFB.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
13-06-2005	**Master's Thesis**	Jun 2003 – Nov 2004

4. TITLE AND SUBTITLE

Modeling Adaptive Middleware and Its Applications to Military Tactical Datalinks

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Lawson, Jason, T., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Bldg 641
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCE/ENG/05-08

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFRL/IFTA
Attn: Mr. Kenneth Littlejohn
Bldg 620, Rm N3-F22
2241 Avionics Circle
WPAFB OH 45433 DSN: 785-6548x3587

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Open systems solutions and techniques have become the de facto standard for achieving interoperability between disparate, large-scale, legacy software systems. A key technology among open systems solutions and techniques is middleware. Middleware, in general, is used to isolate applications from dependencies introduced by hardware, operating systems, and other low-level aspects of system architectures. While middleware approaches are or will be integrated into operational military systems, many open questions exist about the appropriate areas to applying middleware.

Adaptive middleware is middleware that provides an application with a run-time adaptation strategy, based upon system-level interfaces and properties. Adaptive middleware is an example of an active applied research area. Adaptive middleware is being developed and applied to meet the ever-increasing challenges set forth by the next generation of mission-critical distributed real-time and embedded (DRE) systems. The driving force behind many next-generation DRE systems is the establishment of QoS requirements typically associated with workloads that vary dynamically.

The Weapon System Open Architecture (WSOA), an adaptive middleware platform developed by Boeing, is modeled as a part of this research to determine the scalability of the architecture. The WSOA adaptive middleware was previously flight-tested with one tactical node, and the test results represent the performance baseline the architecture. The WSOA adaptive middleware is modeled with 1, 2, 4, 8 and 16 tactical nodes. The results of the modeling and simulation is that the WSOA adaptive middleware can achieve the performance baseline achieved during the original flight-test, in the cases of 1, 2, and 4 tactical nodes. In addition, the results of the modeling and simulation also demonstrate that the WSOA adaptive middleware cannot achieve the original performance baseline, in the cases of 8 and 16 tactical nodes.

15. SUBJECT TERMS
Middleware, Adaptive Middleware, QoS, Quality of Service, Military Tactical Datalinks, Link-16, Open Systems, Distributed Real-Time, Embedded, OPNET®

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
REPORT	**ABSTRACT**	**c. THIS PAGE**			Richard A. Raines, Dr., (ENG)
U	**U**	**U**	**UU**	100	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565, ext 4278; e-mail: Richard.Raines@afit.edu